P86 - CLOSE' Format should be CLOSE file-name.

P96 - OPEN " " " OPEN { } .

DI = Duplicate ID

MO = Memory Overflow.

SCK Routine screwed up decode calls for 9, routine
present. 4 bytes.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICRO-COBOL
AN IMPLEMENTATION OF
NAVY STANDARD HYPO-COBOL
FOR A MICROPROCESSOR-BASED COMPUTER SYSTEM

by

Alan Scott Craig

March 1977

Thesis Advisor: Gary A. Kildall

Approved for public release; distribution unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED
MICRO-COBOL an implementation of Navy Standard Hypo-Cobol for a microprocessor-based computer system	Masters Thesis; March 1977
	6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s)
Alan Scott Craig	

9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Naval Postgraduate School Monterey, California 93940	

11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE
Naval Postgraduate School Monterey, California 93940	March 1977
	13. NUMBER OF PAGES
	170

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
Naval Postgraduate School Monterey, California 93940	Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COBOL, compiler, formal grammar, microprocessor, microcomputer, LALR(1), HYPO-COBOL

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A compiler for ADPESO standard HYPO-COBOL has been implemented on a microcomputer. The implementation provides nucleus level constructs and file options from the ANSII COBOL package along with the PERFORM UNTIL construct from a higher level to give increased structural control. The language was implemented through a self-hosted compiler and run-time package

DD FORM 1473 EDITION OF 1 NOV 68 IS OBSOLETE
1 JAN 73
(Page 1) S/N 0102-014-6601 |

on an 8080 microcomputer-based system. Both compiler and
interpreter can be executed in 12K bytes of user storage.

MICRO-COBOL
an implementation of
Navy Standard Hypo-Cobol
for a microprocessor-based computer system

by

Alan Scott Craig
Captain, United States Marine Corps
B.S., Brigham Young University, May 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1977

ABSTRACT

A compiler for ADPESO standard HYPO-COBOL has been im-
plemented on a microcomputer. The implementation provides
nucleus level constructs and file options from the ANSII
COBOL package along with the PERFORM UNTIL construct from a
higher level to give increased structural control. The
language was implemented through a self-hosted compiler and
run-time package on an 8080 microcomputer-based system.
Both compiler and interpreter can be executed in 12K bytes
of user storage.

CONTENTS

5

I. INTRODUCTION

A. HISTORY OF COBOL

As indicated in the name, COBOL - COmmon Business Oriented Language - was intended to be a common standard computer programming language with consistent implementations on various machines. Backed heavily by the Department of Defense, COBOL has become a widely accepted language for data processing applications. Over the fifteen years of its existence the language has undergone several revisions and still continues to be upgraded and changed [1].

The evolution of COBOL has resulted in a large language containing numerous capabilities, many of which are not appropriate for a given machine nor desired by a class of users. For this reason the COBOL language is broken down into modules which may be implemented at various levels. The minimal standard COBOL, as currently defined, contains only the lowest levels of three modules out of the possible twelve modules which currently exist.

B. MOTIVATIONS OF HYPO-COBOL

None of the existing standard sets of COBOL modules fit
the requirements of the Department of the Navy, and thus
HYPO-COBOL was developed. Rather than taking one of the im-
plementation levels described in the standard, another sub-
set of the complete instruction set was developed which in-
cludes only parts of modules. HYPO-COBOL was designed to
impose minimal requirements on a system for compiler sup-
port. Where possible, short constructs were used in the
place of longer ones. Where multiple reserved words serve
the same function in COBOL, the shortest form was used.
There is no optional verbage in the language, and there are
no duplicate constructs performing the same function.

Limits were placed on all statements that have a vari-
able input format so that all statements have a fixed max-
imum length. Where possible, such constructs were removed
completely from the language. In addition, user defined
names were limited to twelve characters to reduce symbol
table storage requirements.

Rather than include the standard levels of implementa-
tion for all of the modules, constructs were included only
as required. In addition to low level constructs, the PER-
FORM UNTIL construct was included to allow better program
structure. Further justification for the manner of subset-
ting and a highly detailed description of each element of
the language is contained in the HYPO-COBOL Manual [10].

C. MICROCOMPUTERS

Current technological advances in the design of in-
tegrated computer components have lead to the proliferation
of single chip central processors known as microcomputers.
The number of chips produced and the varying capabilities of
each product make generalizations very difficult. The term
microcomputer, however, is generally used to describe a sys-
tem built around one of these processors. Such a system
would have memory, input and output capabilities, and timing
circuts as well as a central processor. One chip systems
with all of these capabilities are currently becoming avail-
able.

1. Hardware

The most significant factor in the proliferation of
microcomputer-based systems has been their cost. Reasonably
powerful central processors can currently be purchased for
less than twenty dollars, resulting in the appearance of
many new applications. Along with the low cost of the cen-
tral processor have come low cost peripheral devices that
are well suited to the speeds and capabilities of the micro-
computers. In the case of traditional users of computers,
the low cost of microcomputer hardware has led to new uses
and to distributed processor networks. Changes in the cost
and capabilities of microcomputers have been dramatic over
the last several years, with more and more capabilities be-
ing offered at lower prices.

2. Software

Software has lagged far behind the developments in hardware for microcomputers. Most of the currently available systems do not support high level languages at all, and where supported, the languages are often systems languages rather than applications oriented languages. One of the restrictions imposed by many high level languages has been the requirement for cross-compiling on a more powerful machine [7]. In addition, some of the resident compilers require large amounts of memory. Recent work on versions of BASIC however, has led to quality resident compilers for scientific type calculations [6].

To allow the use of microprocessor systems in many of the proposed applications, languages need to be developed that will run on microcomputers without placing unreasonable demands on their capabilities and size. If the developments in hardware continue at their present rate, software will almost certainly continue to lag behind. However, current compiler construction techniques do seem to make it possible to provide the required languages, at least on the current types of hardware [3].

D. OBJECTIVES OF MICRO-COBOL

The major objective of this project was to implement HYPO-COBOL on an 8080 microcomputer-based system. As steps toward that objective, the following underlying goals were established: first, define HYPO-COBOL as an LALR(1) grammar [12]. Second, construct a compiler based on a table-driven parser for that LALR(1) grammar. Third, implement an interpreter to run the intermediate language instructions produced by the compiler.

While it was recognized that there would be difficulties in displaying the complete capabilities of the HYPO-COBOL language on the equipment currently available at the Naval Postgraduate School, it was considered feasible to implement a major portion of the subset with the current equipment and software.

One of the justifications for this project was the current standard policy of the Department of Defense to require all computers used in non-tactical environments to be capable of executing COBOL. In the case of the Department of the Navy, the standard that would need to be met for a microcomputer-based system is HYPO-COBOL.

Finally, it should be noted that there was no attempt to add to the HYPO-COBOL definition. One area of investigation was to test the feasibility of the subset. In defining the grammar, areas were found where additions could have been made, and future users may require enhanced capabilities to

make the language fit their requirements. Indications have been made, in the following sections, of places where changes seemed appropriate.

II. MICRO-COBOL MACHINE

A. GENERAL DESCRIPTION

The following sections describe the MICRO-COBOL pseudo-machine architecture in terms of allocated memory areas and pseudo-machine operations. The pseudo machine was the target machine for the compiler and was implemented through a programmed interpretation. The MICRO-COBOL machine has been given first, since all other system components can be described in terms of the target machine.

There were several ways to design the pseudo machine. The parser used produces operations in the order convenient for a stack machine, and other applications have used a simulation of a stack machine to interpret the output of the compiler [6]. The operations required for HYPO-COBOL did not require the use of a stack but could be designed as relatively independent operations. It would be possible to produce an interpreter that consisted of a set of subroutines which would be called directly by machine level operations on the 8080. The emitted code would then consist of instructions to load parameters and calls to the subroutines. This second idea was rejected due to the limited time available for the production of the project and because the code generation would then be very closely tied to the exact implementation of the interpreter. It was de-

13

cided to produce output code for a pseudo machine that would be defined to have all of the needed operations as basic instructions. The machine operators chosen contain all of the information required to perform one complete action required by the language.

The machine contains multiple parameter operators and a program counter that addresses the next instruction to be executed. Three registers are provided which hold eighteen digit numbers used for arithmetic operations along with a subscript stack that is used to compute subscript locations along with a set of flags that are used to pass branching information from one instruction to another.

Addresses in the machine are represented by 16 bit values. Any memory address greater than 20 hexadecimal is valid. Addresses less than 20 hexadecimal will be interpreted as having special significance. For example, addresses one through eight are reserved for subscript stack references. All other addresses in the machine are absolute addresses.

The arithmetic registers allow for the manipulation of signed numbers of up to eighteen decimal digits in length. Included in their representation is a sign indicator and the position of the assumed decimal point for the currently loaded number. While the form of the representation is not specified in the HYPO-COBOL document, it is necessary that there be no loss of precision for operations on numbers hav-

14

ing a full eighteen digits of significance.

There are two major types of numbers defined in the machine. The first is numbers in the DISPLAY mode. These numbers are represented in memory in the standard information exchange code for the peripherals. For microcomputers, the common representation would be in ASCII characters. These numbers may have separate signs indicated by "+" and "-" or may have a "zone" indicator added, denoting a negative sign. Packed decimal format is also available with numbers carried as sequential digit pairs stored in memory. The sign is indicated in the right-most position.

The following flags exist in the machine and can be checked by the instructions for a true or false value: BRANCH flag -- indicates if a branch is to be taken; END OF RECORD flag -- indicates that an end of input condition has been reached when an attempt was made to read input; OVER-FLOW flag -- indicates the loss of information from a register due to a number exceeding the available size; INVALID flag -- indicates an invalid action in writing to a direct access storage device.

The following resources are required for a minimal implementation of this machine: a system input device capable of receiving low volume input, a system output device capable of displaying low volume output, and a direct access storage device capable of storing, reading, and writing files and programs.

B. MEMORY ORGANIZATION

Memory is divided into three major sections: (1) the data areas defined by the DATA DIVISION statements, (2) the code area, (3) and the constants area. No particular order of these sections is required. The first two areas assume the ability to both read and write, but the third only requires the ability to be read.

The data area contains variables defined by the DATA DIVISION statements, constants set in the WORKING STORAGE SECTION, and all file control blocks and buffers. These elements will be manipulated by the machine in accordance with the code instructions.

C. MACHINE OPERATIONS

1. Format

All of the machine operations consist of an operation number followed by a list of parameters. The sections that follow describe the various instructions, list the required parameters, and describe the actions taken by the machine in executing each instruction. As each instruction is fetched from memory, the program counter automatically increments by one.

2. Arithmetic operations

There are five arithmetic instructions which act only on the registers. In all cases, the result is placed

in register two. Operations are allowed to destroy the in-
put values during the process of creating a result. There-
fore, a number loaded into a register will not be available
for a subsequent operation.

ADD: (addition). Sum the contents of register zero
and register one.
Parameters: no parameters are required.

SUB: (subtract). Subtract register one from register
zero.
Parameters: no parameters are required.

MUL: (multiply). Multiply register zero by register
one.
Parameters: no parameters are required.

DIV: (divide). Divide register zero by the value in
register one. The remainder is not retained.
Parameters: no parameters are required

RND: (round). Round register two to the last signi-
ficant decimal place.
Parameters: no parameters are required.

3. Branching

All of the branching instructions are accomplished
by changing the value of the program counter. Some are ab-
solute branches and some test for condition flags that are
set by the other instructions. Branches may also test the

state of the registers or perform direct comparisons on memory fields.

Several instructions use the same conditional branching conventions. First, the branch flag is checked for its current setting. If it is true, then a branch is made by changing the program counter to the value of the <branch address>. The branch flag is then set to false. If the flag was originally false, the program counter is incremented to the next sequential instruction.

BRN: (branch to an address). Load the program counter with the <branch address>.
Parameters: <branch address>

The next three instructions share a common format. The memory field addressed by the <memory address> is checked for the <address length>, and if all the characters match the test condition, then the branch flag is complemented. A conditional branch is taken after the test.
Parameters: <memory address> <address length> <branch address>

CAL: (compare alphabetic). Compare a memory field for alphabetic characters.

CNS: (compare numeric signed). Compare a field for numeric characters allowing for a sign character.

CNU: (compare numeric unsigned). Compare a field for numeric characters only.

DEC: (decrement a count and branch if zero). Decrement the value of the <address counter> by one, and if the result is zero, the program counter is set to the address given. If the result is not zero, then the program counter is incremented by four. If the result is zero before decrementing, the branch is taken.
Parameters: <address counter> <branch address>

EOR: (branch on end of records flag). If the end-of-records flag is true, it is set to false and the program counter is set to the <branch address>. If false, the program counter is incremented by two.
Parameters: <branch adress>

GDP: (go to - depending on). The memory location addressed by the <number adress> is read for the number of bytes indicated by the <memory length>. This number indicates which of the <branch addresses> is to be used. The first parameter is a bound on the number of branch addresses. If the number is within the range, the program counter is set to the indicated address. An out of bounds value causes the program counter to be advanced to the next sequential instruction.
Parameters: <bound number - byte> <memory length> <memory address> <branch addr-1> <branch addr-2> ... <branch addr-n>

INV: (branch if invalid-file-action flag true). If the invalid-file-action flag is true, then it is set to false, and the program counter is set to the branch ad-

dress. If it is false, the program counter is incremented by two.

Parameters: <branch address>

PER: (perform). The code address pointed to by the <change address> is loaded with the value of the <return address>. The program counter is then set to the <branch address>.

Parameters: <branch address> <change address> <return address>

RET: (return). If the value of the <branch address> is not zero, then the program counter is set to its value, and the <branch address> is set to zero. If the <branch address> is zero, the program counter is incremented by two.

Parameters: <branch address>

REU: (register equal). This instruction checks for a zero value in register two. If it is zero, the branch flag is complemented. A conditional branch is taken.

Parameters: <branch address>

RGT: (register greater than). Register two is checked for a negative sign. If present, the branch flag is complemented. A conditional branch is taken.

Parameters: <branch address>

RLT: (register less than). Register two is checked for a positive sign, and if present, the branch flag is complemented. A conditional branch is taken.

Parameters: <branch address>

SER: (branch on size error). If the overflow flag is
true, then the program counter is set to the branch address,
and the overflow flag is set to false. If it is false, then
the program counter is incremented by two.

Parameters: <branch address>

The next three instructions all perform the same
function and have the same general format. They compare two
strings and perform a conditional branch. If the test con-
dition is true, the branch flag is complemented prior to
taking the conditional branch.

Parameters: <string addr-1> <string addr-2> <length - ad-
dress> <branch address>

SEQ: (strings equal). Compare two string for equal
characters.

SGT: (string greater than). Compare string one for
greater than string two.

SLT: (string less than). Compare string one for less
than string two.

4. Moves

The machine supports a variety of move operations
for various formats and types of data. It does not support
direct moves of numeric data from one memory field to anoth-
er. Instead, all of the numeric moves go through the regis-

21

ters. This greatly reduced the number of instructions since all of the numeric types need to be supported by moves into and out of the registers for arithmetic operations.

The next seven instructions all perform the same function. They load a register with a numeric value and differ only in the type of number that they expect to see in memory at the <number address>. All seven cause the program counter to be incremented by five. Their common format is given below.

Parameters: <number address> <byte length> <byte decimal count> <byte register to load>

LD0: (load a numeric literal). Note that the decimal point indicator is not set in this instruction format. The literal will have an actual decimal point in it if required.

LD1: (load a numeric field).

LD2: (load a numeric field with an internal trailing sign).

LD3: (load a numeric field with an internal leading sign).

LD4: (load a numeric field with a separate leading sign).

LD5: (load a numeric field with a separate trailing sign).

LD6: (load a packed numeric field).

MED: (move into a alphanumeric edited field). The
edit mask is loaded into the <to address> to set up the
move, and then the <from address> information is loaded. The
program counter is incremented by ten.

Parameters: <to address> <from address> <length of move>
<edit mask address> <edit mask length>

MNE: (move into a numeric edited field). First the
edit mask is loaded into the receiving field, and then the
information is loaded. Any decimal point alignment required
will be performed. If truncation of significant digits is a
side effect, the overflow flag is not set. The program
counter is incremented by twelve.

Parameters: <to address> <from address> <address length of
move> <edit mask address> <address mask length> <byte to de-
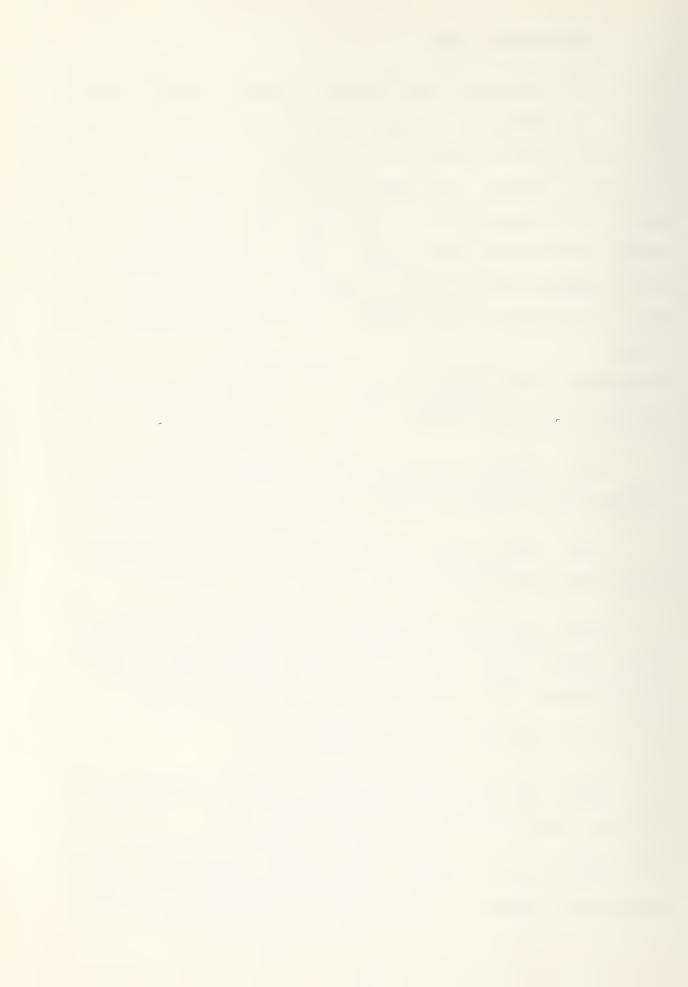cimal count> <byte from decimal count>

MOV: (move into an alphanumeric field). The memory
field given by the <to address> is filled by the from field
for the <move length> and then filled with blanks in the
following positions for the <fill count>.

Parameters: <to address> <from address> <address move
length> <address fill count>

SII: (store immediate register two). The contents of
register two are stored into register zero and the decimal
count and sign are indicators set.

Parameters: none.

The store instructions are grouped in the same order as the load instructions. Register two is stored into memory at the indicated location. Any alignment is performed, and if a non-zero leading digit is truncated by the operation, the overflow flag is set. All five of the store instructions cause the program counter to be incremented by four. The format for these instructions is as follows. Parameters: <address to store into> <byte length> <byte decimal count>

ST0: (store into a numeric field).

ST1: (store into a numeric field with an internal trailing sign).

ST2: (store into a numeric field with an internal leading sign).

ST3: (store into a numeric field with a separate trailing sign).

ST4: (store into a numeric field with a separate leading sign).

ST5: (store into a packed numeric field).

5. Input-output

The following instructions perform input and output operations. The required operations are specified in the HYPO-COBOL manual, but the exact definitions of file formats and access methods are not defined. Files in this machine

24

are defined as having the following characteristics: they
are either sequential or random, and, in general, files
created in one mode are not required to be readable in the
other mode. Standard files consist of fixed length records,
and variable length files need not be readable in a random
mode. Further, there must be some character or charcter
string that delimits a variable length record.

ACC: (accept). Read from the system input device into
memory at the location given by the <memory address>. The
program counter is incremented by three.
Parameters: <memory address> <byte length of read>

CLS: (close). Close the file whose file control block
is addressed by the <fcb address>. The program counter is
incremented by two.
Parameters: <fcb address>

DIS: (display). Print the contents of the data field
pointed to by <memory address> on the system output device
for the indicated length. The program counter is increment-
ed by three.
Parameters: <memory address> <byte length>

There are three open instructions with the same for-
mat. In each case, the file defined by the file control
block referenced will be opened for the mode indicated. The
program counter is incremented by two.
Parameters: <fcb address>

OPN: (open a file for input).

OP1: (open a file for output).

OP2: (open a file for both input and output). This is only valid for files on a random access device.

The following file actions all share the same format. Each performs a file action on the file referenced by the file control block. The record to be acted upon is given by the <record address>. The program counter is incremented by six.

Parameters: <fcb address> <record address> <record length - address>

DLS: (delete a record from a sequential file). Remove the record that was just read from the file. The file is required to be open in the input-output mode.

RDF: (read a sequential file). Read the next record into the memory area.

WIF: (write a record to a sequential file). Append a new record to the file.

RVL: (read a variable length record).

WVL: (write a variable length record).

RWS: (rewrite sequential). The rewrite operation writes a record from memory to the file, overlaying the last record that was read from the device. The file must be open

in the input-output mode.

The following file actions require random files
rather than sequential files. They all make use of a random
file pointer which consists of a <relative address> and a
<relative length>. The memory field holds the number to be
used in disk operations or contains the relative record
number of the last disk action. The relative record number
is the record count on the file starting with one. After
the file action, the program counter is incremented by
nine.

Parameters: <fcb address> <record address> <record length -
address> <relative address> <relative length - byte>.

DLR: (delete a random record). Delete the record ad-
dressed by the relative record number.

RRR: (read random relative). Read a random record
relative to the record number.

RRS: (read random sequential). Read the next sequen-
tial record from a random file. The relative record number
of the record read is loaded into the memory reference.

RWR: (rewrite a random record).

WRR: (write random relative). Write a record into
the area indicated by the memory reference.

WRS: (write random sequential). Write the next
sequential record to a random file. The relative record

27

number is returned.

6. Special instructions

The remaining instructions perform special functions
required by the machine that do not relate to any of the
previous groups.

NOT: (negitive test). Negate the value of the branch
flag.
Parameters: no parameters are required.

LDI: (load a code address direct). Load the <code
address> with the number indicated by the <memory address>.
Parameters: <code address> <memory address> <length - byte>

SCR: (calculate a subscript). Load the subscript
stack with the value indicated from memory. The address
loaded into the stack is the <initial address> plus an
offset. Multiplying the <field length> by the number in the
<memory reference> gives the offset value.
Parameters: <initial address> <field length> <memory refer-
ence> <memory length> <stack level>

SID: (stop with display). Display the indicated in-
formation and then stop.
Parameters: <memory address> <length - byte>

SIP: (stop). terminate the actions of the machine.
Parameters: no parameters are required.

The following instructions are used in setting up the machine environment and cannot be used in the normal execution of the machine.

BST: (backstuff). Resolve a reference to a label. Labels may be referenced prior to their definition, requiring a chain of resolution addresses to be maintained in the code. The latest location to be resolved is maintained in the symbol table and a pointer at that location indicates the next previous change. A zero pointer indicates no prior occurrences of the label. The code address referenced by <change address> is examined and if it contains zero, it is loaded with the <new address>. If it is not zero, then the contents are saved, and the process is repeated with the saved value as the change address after loading the <new address>.
Parameters: <change address> <new address>

INT: (initialize memory). Load memory with the <input string> for the given length at the <memory address>.
Parameters: <memory address> <address length> <input string>

SCD: (start code). Set the initial value of the program counter.
Parameters: <start address>

TER: (terminate). Terminate the initialization process and start executing code.
Parameters: no parameters are required.

29

III. MICRO-COBOL IMPLEMENTATION

A. COMPILER IMPLEMENTATION

1. General method

The LALR parser-table construction programs used here are based on the work of Knuth [9]. His work defines two methods of testing a grammar to see if it is LR(k). One of these methods leads to the creation of a set of tables that can be used to drive the parse actions of a compiler. While difficult to implement in the form given by Knuth, the method has been developed in usable form for subsets of the grammars that are LR(k). References 2 and 3 contain detailed discussions of the methods currently available. The algorithm used to develop the tables for the MICRO-COBOL compiler was developed by W. Lalonde [12].

The compiler was designed to read the source language statements from a diskette or other mass storage device, extract the needed information for the symbol table, and write the output code back onto the diskette all in one pass of the source program. The grammar was initially defined for the entire language, but the size constraints placed on the implementation required smaller tables. The grammar was then defined in two parts which run in succession. The major method of passing information from the

30

first part to the second is by placing the information in the symbol table.

The output code from the compiler consists of the operations that have been previously defined. They were designed as an intermediate language that would be executed by the interpreter described in section B. The vast differences between the operations available for the target computer and the operations necessary to support COBOL made this approach easier than 8080 machine code.

2. Control flow

The compiler has been designed so that the operation of the two parts would be transparent to the user. When the first part is loaded it brings in with its code a reader program which loads the second file automatically. Prior to calling the reader program, the first part writes any pending code to the disk and loads all toggles to a common area ready to be read by the second part.

Internally, the control of the two parts is identical. The parser is called after initialization and runs until it either finishes its task or reaches an unrecoverable error state. The major subroutines in the compiler are the scanner and the production case statement. Both are controlled in their actions by the parser.

3. Internal structures

The major internal structure is the symbol table. It was designed as a list where the elements in the list are the descriptions of the various symbols in the program. As new symbols are encountered they are added to the end of the list. Symbols already in the list can be accessed through the use of a "current symbol pointer." The location of items in the list is determined by checking the identifier against a hash table that points to the first entry in the symbol table with that hash code. A chain of collision addresses is maintained in the symbol table which links entries which have the same hash value.

All of the items in the symbol table contain the following information: a collision field, a type field, the length of the identifier, and the address of the item. If an item in the symbol table is a data field, the following information is included in the table: the length of the item, the level of the data field, an optional decimal count, an optional multiple occurrence count, and the address of the edit field, if required. If the item is a file name then the following additional information is included: the file record length, the file control block address, and the optional symbol table location of the relative record pointer. If the item is a label, then the only additional information is the location of the return instruction at the end of the paragraph or section.

In addition to the symbol table, two stacks are used for storing information: the level stack and the identifier stack. In both cases, they are used to hold pointers to entries in the symbol table. The identifier stack is used to collect multiple occurrences in such statements as the GO TO - DEPENDING statement. The level stack is used to hold information about the various levels that make up a record description.

The parser has control of a set of stacks that are used in the manipulation of the parse states. In addition to the state stack that is required by the parser, part one has a value stack and part two has two different value stacks that operate in parallel with the parser state stack. The use of these stacks is described below.

4. Part one

The first part of the compiler is primarily concerned with building the symbol table that will be used by the second part. The actions corresponding to each parse step are explained in the sections that follow. In each case, the grammar rule that is being applied is given, and an explanation of what program actions take place for that step has been included. In describing the actions taken for each parse step there has been no attempt to describe how the symbol table is constructed or how the values are preserved on the stack. The intent of this section is to describe what information needs to be retained and at what

33

point in the parse it can be determined. where no action is required for a given statement, or where the only action is to save the contents of the top of the stack, no explaination is given. Questions regarding the actual manipulation of information should be resolved by consulting the programs.

1 <program> ::= <id-div> <e-div> <d-div> PROCEDURE

Reading the word PROCEDURE terminates the first part of the compiler.

2 <id-div> ::= IDENTIFICATION DIVISION. PROGRAM-ID.
                <comment> . <auth> <date> <sec>

3 <auth> ::= AUTHOR . <comment> .

4   ¦ <empty>

5 <date> ::= DATE-WRITTEN . <comment> .

6   ¦ <empty>

7 <sec> ::= SECURITY . <comment> .

8   ¦ <empty>

9 <comment> ::= <input>

10   ¦ <comment> <input>

11 <e-div> ::= ENVIRONMENT DIVISION . CONFIGURATION SECTION.
                <scr-obj> <i-o>

12 <src-obj> ::= SOURCE-COMPUTER . <comment> <debug> .
                OBJECT-COMPUTER . <comment> .

13 <debug> ::= DEBUGGING MODE

Set a scanner toggle so that debug lines will be read.

14   ¦ <empty>

15 <i-o> ::= INPUT-OUTPUT SECTION . FILE-CONTROL .

34

```
                    <file-control-list> <ic>
16              ¦ <empty>
17  <file-control-list> ::= <file-control-entry>
18                          ¦ <file-control-list> <file-control-entry>
19  <file-control-entry> ::= SELECT <id> <attribute-list> .
```

At this point all of the information about the file
has been collected and the type of the file can be
determined. File attributes are checked for compata-
bility and entered in the symbol table.

```
20  <attribute-list> ::= <one attrib>
21                       ¦ <attribute-list> <one attrib>
22  <one-attrib> ::= ORGANIZATION <org-type>
23               ¦ ACCESS <acc-type> <relative>
24               ¦ ASSIGN <input>
```

A file conrol block is built for the file using an INT
operator.

```
25  <org-type> ::= SEQUENTIAL
```

No information needs to be stored since the default
file organization is sequential.

```
26              ¦ RELATIVE
```

The relative attribute is saved for production 19.

```
27  <acc-type> ::= SEQUENTIAL
```

This is the default.

```
28              ¦ RANDOM
```

The random access mode needs to be saved for produc-
tion 19.

```
29  <relative> ::= RELATIVE <id>
```

The pointer to the identifier will be retained by the

current symbol pointer, so this production only saves
a flag on the stack indicating that the production did
occur.

30   ¦ <empty>

31   <ic> ::= I-O-CONTROL . <same-list>

32   ¦ <empty>

33   <same-list> ::= <same-element>

34   ¦ <same-list> <same-element>

35   <same-element> ::= SAME <id-string> .

36   <id-string> ::= <id>

37   ¦ <id-string> <id>

38   <d-div> ::= DATA DIVISION . <file-section> <work> <link>

39   <file-section> ::= FILE SECTION . <file-list>

Actions will differ in production 64 depending upon
whether this production has been completed. A flag
needs to be set to indicate completion of the file
section.

40   ¦ <empty>

The flag, indicated in production 39, is set.

41   <file-list> ::= <file-element>

42   ¦ <file-list> <file-element>

43   <files> ::= FD <ic> <file-control> . <record-description>

This statement indicates the end of a record descrip-
tion, and the length of the record and its address can
now be loaded into the symbol table for the file
name.

44   <file-control> ::= <file-list>

45   ¦ <empty>

```
46   <file-list> ::= <file-element>

47                 ¦ <file-list> <file-element>

48   <file-element> ::= BLOCK <integer> RECORDS

49                 ¦ RECORD <rec-count>

     The record length can be saved for comparison with the

     calculated length from the picture clauses.

50                 ¦ LABEL RECORDS STANDARD

51                 ¦ LABEL RECORDS OMITTED

52                 ¦ VALUE OF <id-string>

53   <rec-count> ::= <integer>

54                 ¦ <integer> TO <integer>

     The TO option is the only indication that the file

     will be variable length. The maximum length must be

     saved.

55   <work> ::= WORKING-STORAGE SECTION . <record-description>

56           ¦ <empty>

57   <link> ::= LINKAGE SECTION . <record-description>

58           ¦ <empty>

59   <record-description> ::= <level-entry>

60                         ¦ <record-description> <level-entry>

61   <level-entry> ::= <integer> <data-id> <redefines>

                       <data-type> .
```

The level entry needs to be loaded into the level
stack. The level stack is used to keep track of the
nesting of field definitions in a record. At this
time there may be no information about the length of
the item being defined, and its attributes may depend
entirely upon its constituent fields. If there is a

37

pending literal, the stack level to which it applies
is saved.

62 <data-id> ::= <id>

63 | FILLER

An entry is built in the symbol table to record infor-
mation about this record field. It cannot be used ex-
plicitly in a program because it has no name, but its
attributes will need to be stored as part of the total
record.

64 <redefines> ::= REDEFINES <id>

The redefines option gives new attributes to a previ-
ously defined record area. The symbol table pointer
to the area being redefined is saved so that informa-
tion can be transfered from one entry to the other.
In addition to the information saved relative to the
redefinition, it is necessary to check to see if the
current level number is less than or equal to the lev-
el recorded on the top of the level stack. If this is
true, then all information for the item on the top of
the stack has been saved and the stack can be re-
duced.

65 | <empty>

As in production 64, the stack is checked to see if
the current level number indicates a reduction of the
level stack. In addition, special action needs to be
taken if the new level is 01. If an 01 level is en-
countered at this production prior to production 39 or
40 (the end of the file area), it is an implied rede-

38

finition of the previous 01 level. In the working
storage section, it indicates the start of a new
record.

66 <data-type> ::= <prop-list>

67 | <empty>

68 <prop-list> ::= <data-element>

69 | <prop-list> <data-element>

70 <data-element> ::= PIC <input>

The <input> at this point is the character string that
defines the record field. It is analyzed and the ex-
tracted information is stored in the symbol table.

71 | USAGE COMP

The field is defined to be a packed numeric field.

72 | USAGE DISPLAY

The DISPLAY format is the default, and thus no special
action occurs.

73 | SIGN LEADING <separate>

This production indicates the presence of a sign in a
numeric field. The sign will be in a leading posi-
tion. If the <separate> indicator is true, then the
length will be one longer than the picture clause, and
the type will be changed.

74 | SIGN TRAILING <separate>

The same information required by production 73 must be
recorded, but in this case the sign is trailing rather
than leading.

75 | OCCURS <integer>

The type must be set to indicate multiple occurrences,

and the number of occurrences saved for computing the
space defined by this field.

76 | SYNC <direction>

Syncronization with a natural boundary is not required
by this machine.

77 | VALUE <literal>

The field being defined will be assigned an initial
value determined by the value of the literal through
the use of an INT operator. This is only valid in the
WORKING-STORAGE SECTION.

78   <direction> ::= LEFT

79 | RIGHT

80 | <empty>

81   <separate> ::= SEPARATE

The separate sign indicator is set on.

82 | <empty>

83 <literal> ::= <input>

The input string is checked to see if it is a valid
numeric literal, and if valid, it is stored to be used
in a value assignment.

84 | <lit>

This literal is a quoted string.

85 | ZERO

As is the case of all literals, the fact that there is
a pending literal needs to be saved. In this case and
the three following cases, an indicator of which
literal constant is being saved is all that is re-
quired. The literal value can be reconstructed

later.

86 | SPACE

87 | QUOTE

88   <integer> ::= <input>

     The input string is converted to an integer value for

     later internal use.

89   <id> ::= <input>

     The input string is the name of an identifier and is

     checked aginst the symbol table. If it is in the sym-

     bol table, then a pointer to the entry is saved.   If

     it is not in the symbol table, then an entry is added

     and the address of that entry is saved.

     5. Part two

         The second part includes all of the PROCEDURE DIVI-

SION, and is the part where code generation takes place. As

in the case of the first part, there was no intent to show

how various pieces of information were retrieved but only

what information was used in producing the output code.

  1 <p-div> ::= PROCEDURE DIVISION <using> .

                   <proc-body> END .

     This production indicates termination of the compila-

     tion.   If the program has sections, then it will be

     necessary to terminate the last section with a REI 0

     instruction.   The code will be ended by the output of

     a TER operation.

  2 <using> ::= USING <id-string>

3 | <empty>

4 <id-string> ::= <id>

The identifier stack is cleared and the symbol table
address of the identifier is loaded into the first
stack location.

5 | <id-string> <id>

The identifier stack is incremented and the symbol
table pointer stacked.

6 <proc-body> ::= <paragraph>

7 | <proc-body> <paragraph>

8 <paragraph> ::= <id> . <sentence-list>

The starting and ending address of the paragraph are
entered into the symbol table. A return is emitted as
the last instruction in the paragraph (RET 0). When
the label is resolved, it may be necessary to produce
a BST operation to resolve previous references to the
label.

9 | <id> SECTION .

The starting address for the section is saved. If it
is not the first section, then the previous section
ending address is loaded and a return (RET 0) is out-
put. As in production 8, a BST may be produced.

10 <sentence-list> ::= <sentence>

11 | <sentence-list> <sentence> .

12 <sentence> ::= <imperative>

13 | <conditional>

14 | ENTER <id> <opt-id>

This construct is not implemented. An ENTER allows

42

statements from another language to inserted in the
source code.

15 <imperative> ::= ACCEPT <subid>

ACC <address> <length>

16   ¦ <arithmetic>

17   ¦ CALL <lit> <using>

This is not implemented.

18   ¦ CLOSE <id>

CLS <file control block address>

19   ¦ <file-act>

20   ¦ DISPLAY <lit/id> <opt-lit/id>

The display operator is produced for the first literal
or identifier (DIS <address> <length>). If the second
value exists, the same code is also produced for it.

21   ¦ EXIT <program-id>

RET 0

22   ¦ GO <id>

BRN <address>

23   ¦ GO <id-string> DEPENDING <id>

GDP is output, followed by a number of parameters:
<the number of entries in the identifier stack> <the
length of the depending identifier> <the address of
the depending identifier> <the address of each iden-
tifier in the stack>.

24   ¦ MOVE <lit-id> TO <subid>

The types of the two fields determine the move that is
generated. Numeric moves go through register two us-
ing a load and a store. Non-numeric moves depend upon

45

the result field and may be either MOV, MED or MNE.
Since all of these instructions have long parameter
lists, they have not been listed in detail.

25 | OPEN <type-action> <id>
This produces either OPN, OP1, or OP2 depending upon
the <type-action>. Each of these is followed by a
file control block address.

26 | PERFORM <id> <thru> <finish>
The PER operation is generated followed by the <branch
address> <the address of the return statement to be
set> and <the next instruction address>.

27 | <read-id>

28 | STOP <terminate>
If there is a terminate message, then SPD is produced
followed by <message address> <message length>. Oth-
erwise STP is emitted.

29   <conditional> ::= <arithmetic> <size-error> <imperative>
A BST operator is output to complete the branch around
the imperative from production 65.

30 | <file-act> <invalid> <imperative>
A BST operator is output to complete the branch from
production 64.

31 | IF <condition> <action> ELSE <imperative>
Two BST operators are required. The first fills in
the branch to the ELSE action. The second completes
the branch around the <imperative>.

32 | <read-id> <special> <imperative>
A BST is produced to complete the branch around the

44

<imperative>.

33 <Arithmetic> ::= ADD <1/id> <opt-1/id> TO <subid> <round>

The existence of multiple load and store instructions
make it difficult to indicate exactly what code will
be generated for any of the arithmetic instructions.
The type of load and store will depend on the nature
of the number involved, and in each case the standard
parameters will be produced. This parse step will in-
volve the following actions: first, a load will be em-
itted for the first number into regster zero. If
there is a second number, then a load into register
one will be produced for it, followed by an ADD and a
STI. Next a load into register one will be generated
for the result number. Then an ADD instruction will
be emitted. Finally, if the round indicator is set, a
RND operator will be produced prior to the store.

34 | DIVIDE <1/id> INTO <subid> <round>

The first number is loaded into register zero. The
second operand is loaded into register one. A DIV
operator is produced, followed by a RND operator prior
to the store, if required.

35 | MULTIPLY <1/id> BY <subid> <round>

The multiply is the same as the divide except that a
MUL is produced.

36 | SUBTRACT <1/id> <opt-1/id> FROM

                          <subid> <round>

Subtaction generates the same code as the ADD except
that a SUB is produced in place of the last ADD.

45

37 <file-act> ::= DELETE <id>

Either a DLS or a DLR will be produced along with the required parameters.

38 | REWRITE <id>

Either a RWS or a RWR is emitted, followed by parameters.

39 | WRITE <id> <special-act>

There are four possible write instructions: WTF, WVL, WRS, and WRR.

40 <condition> ::= <lit/id> <not> <cond-type>

One of the compare instructions is produced. They are CAL, CNS, CNU, RGT, RLT, REQ, SGT, SLT, and SEQ. Two load instructions and a SUB will also be emitted if one of the register comparisons is required.

41 <cond-type> ::= NUMERIC

42 | ALPHABETIC

43 | <compare> <lit/id>

44 <not> ::= NOT

NEG

45 | <empty>

46 <compare> ::= GREATER

47 | LESS

48 | EQUAL

49 <ROUND> ::= ROUNDED

50 | <empty>

51 <terminate> ::= <literal>

52 | RUN

53 <special> ::= <invalid>

54 | END

An ERO operator is produced followed by a zero. The
zero acts as a filler in the code and will be back-
stuffed with a branch address. In this production and
several of the following, there is a forward branch on
a false condition past an imperative action. For an
example of the resolution, examine production 32.

55   <opt-id> ::= <subid>

56 | <empty>

57   <action> ::= <imperative>

    BRN 0

58 | NEXT SENTENCE

    BRN 0

59   <thru> ::= THRU <id>

60 | <empty>

61   <finish> ::= <1/id> TIMES

    LDI <address> <length> DEC 0

62 | UNTIL <condition>

63 | <empty>

64   <invalid> ::= INVALID

    INV 0

65   <size-error> ::= SIZE ERROR

    SER 0

66   <special-act> ::= <when> ADVANCING <how-many>

67 | <empty>

68   <when> ::= BEFORE

69 | AFTER

70   <how-many>::= <integer>

47

71 | PAGE

72 <type-action> ::= INPUT

73 | OUTPUT

74 | I-O

75 <subid> ::= <subscript>

76 | <id>

77 <integer> ::= <input>

    The value of the input string is saved as an internal

    number. •

78 <id> ::= <input>

    The identifier is checked aginst the symbol table, if

    it is not present, it is entered as an unresolved la-

    bel.

79 <1/id> ::= <input>

    The input value may be a numeric literal. If so, it

    is placed in the constant area with an INT operand.

    If it is not a numeric literal, then it must be an

    identifier, and it is located in the symbol table.

80 | <subscript>

81 | ZERO

82 <subscript> ::= <id> (<input>)

    If the identifier was defined with a USING option,

    then the input string is checked to see if it is a

    number or an identifier. If it is an identifier, then

    an SCR operator is produced.

83 <opt-1/id> ::= <1/id>

84 | <empty>

85 <nn-lit> ::= <lit>

48

The literal string is placed into the constant area
using an INT operator.

86   ¦ SPACE

87   ¦ QUOTE

88   <literal> ::= <nn-lit>

89   ¦ <input>

The input value must be a numeric literal to be valid
and is loaded into the constant area using an INT.

90   ¦ ZERO

91   <lit/id> ::= <l/id>

92   ¦ <nn-lit>

93   <opt-lit/id> ::= <lit/id>

94   ¦ <empty>

95   <program-id> ::= <id>

96   ¦ <empty>

97   <read-id> ::= READ <id>

There are four read operations: RDF, RVL, RKS, and
RKR.

The output code file is the only product of the com-
piler that is retained. All of the needed information has
been extracted from the symbol table, and it is not required
by the interpreter. Code will be generated for all programs
including those that contain errors and can be examined
through the use of the decode program. This program
translates the output file into a listing of code operators
followed by the parameters.

B. INTERPRETER IMPLEMENTATION

1. General structure

The format that has been presented for the output code determines the general form of the interpreter. If it had not been possible to transform the instructions from the compiler into a set of call-like commands, it would have been necessary to implement a stack in the interpreter. In general, the interpreter contains a large "case statement" which decodes each operation and either calls subroutines to perform the required actions or acts directly on the run-time environment to control the actions of the interpreter. All communication between instructions is done through common areas in the program where information can be stored for later use.

The design of the interpreter has been modularized in an attempt to allow easy transition to other hardware configurations and operating systems. If desired, any section of the instructions could be implemented in assembly language modules or could be passed to the operating system for action. The entire system has been coded in PL/M for consistency, ease of development, and maximum portability [7].

2. Code modules

The following sections explain the interpreter by noting the specific manner in which the machine instructions

defined in section II-C have been implemented. The divisions are the same as those in section II-C.

a. Arithmetic instructions

Since the machine was defined as having only one set of arithmetic registers, it was necessary to convert all numeric input to one form. The packed decimal format was chosen as the format that would be used in the registers. This conversion process slows down the arithmetic operations slightly, but the reduction of the interpreter memory size was considered more important.

All of the arithmetic operations take place in a set of three work areas or registers. Each of these areas is ten bytes long and can contain an eighteen digit number with one fill character on each end. The extra space facilitates checking for overflow and also makes rounding operations easier. The language does not support the COMPUTE verb, so no storage of intermediate results is required from one instruction to another.

All of the arithmetic instructions use the packed decimal feature of the 8080 as a basis for their actions. Each of the instructions depends on the basic operation of adding two registers: subtraction is accomplished using nines complement arithmetic, multiplication is done through a shift and add algorithm, and division by a shift and subtract method.

If the amount of computations required by a given application make it necessary to speed up these instructions, they could be replaced by a package in assembly language. Extending the grammar to include the COMPUTE verb would require changes in the compiler to allow for temporary locations, but it could be included.

b. Branching

The operation of the interpreter is controlled by a program counter that points to the next operation to be performed. All branching is done by changing the normal sequential order of execution of instructions. In addition to acting directly on the program counter, branching instructions use the branch flag to determine when changes should be made. All of the addresses that point to code are absolute addresses and can be loaded directly into the program counter.

c. Input-output operations

All of the input and output operations use the CP/M interface capabilities [5]. The program expects to see the files in the form that the CP/M editor would have created them. The physical records on the disk are assumed to be 128 bytes in length and have all logical records ending with a carriage-return and a line-feed sequence. There is only one type of file under CP/M, so all restrictions on mixing modes of files are removed for fixed length files. Files created in one program as sequential can be accessed as ran-

52

dom files in another program. Variable length files cannot be accessed in a random fashion because there is no way to compute the starting address of each record.

Where possible, the interface routines have been localized in the programs to simplify transportation to another operating environment. Items relating to file control blocks, disk record lengths, and other system parameters have been established as literals in the programs, rather than entered as numbers, so that changes will not have to be made throughout the code.

a. Moves

As noted previously, the machine lacks numeric moves. There were two major reasons for leaving out the various moves of numeric data. The first was that the added moves would have required more program space, and the second was to simplify the coding and checking of the program. Since all of the numeric types are supported with register load and store operations, any move can be accomplished by a load into register two and a store into the result field.

Alpha-numeric moves are supported as direct moves from memory to memory. If speed is required for a numeric move, the fields concerned can be redefined as alpha-numeric and the memory move used. However, this type of move will only work on two numbers that have exactly the same representation in the computer.

53

Edited moves also are from memory to memory, but they involve several additional steps. The edit mask is loaded into the result field before any characters are loaded, and each character in both the receiving field and the sending field is examined to determine what action should be taken in addition to a move.

3. Limitations

The MICRO-COBOL implementation did not lend itself to support of the Interprogram Communications Module. There was no capability in the operating system to dump the memory image onto the disk or to restore it. It would be possible to implement such a supervisor call, or a one way call could perhaps be implemented from one program to another without the posibility of a return to the calling program. If required by an application where modification of the operating system was not practical, a small overlay program could be written as an independent function to be loaded with the interpreter. If large systems are to be run on microcomputers with minimal memory, some type of interprogram communications would greatly facilitate their design.

C. SOFTWARE TOOLS

As in any software development, one of the things that was most important to the success of this project was the software support for the development effort. This system was developed on the 360/67 rather than on the 8080. Using

the Intel INTERP program [8] and the CP/M simulator
developed by at the Naval Postgraduate School [11], it was
possible to both compile programs on CP/CMS and run the gen-
erated code.   This facility removed the necessity of tran-
sporting code from the 360 to the 8080 for testing and
greatly improved the productivity.

Using the simulator did not result in exactly the same
product as would have been developed if the project had been
done entirely on the 8080.   It was not possible to load a
program on the simulator without destroying the core image
currently in the simulator.   In particular, the first part
of the compiler could not leave the symbol table for the
second part if the second part was loaded by a normal load.
This problem was resolved by writing a set of small programs
that read in the sequence of compiler components from simu-
lated memory image files.   These programs have been included
in this document so that, if future work is done, the simu-
lator could be used again.

IV. CONCLUSIONS

.

This project demonstrates the feasibility of applying modern compiler construction techniques to the implementation of a language developed prior to the work on formal grammars. Not only is it possible to construct a compiler for HYPO-COBOL using an LALR(1) parser, but the resulting programs are highly compact. This allows the implementation of the compiler on smaller machines and increases the number of target systems.

Only a limited number of programs have been written using the compiler, and no attempt has been made to train others in its use. However, adapting to the subset should not be a major problem for a programmer experienced in writing standard COBOL. There have been no extensive timing tests of the system, but current indications are that both the compiler and interpreter operate at an acceptable rate.

There are several areas that could be enhanced in this implementation of HYPO-COBOL. One of these areas is the interprogram communication module. Due to the limitations on core size usually imposed by microcomputer systems, it would be very helpful to be able to compile a set of programs that could be used together as a single module. Several ideas were presented in the body of this paper which indicate how the interprogram communication module could be developed.

The GIVING option for arithmetic statements could be added to the grammar. This option would improve comptational programs, and could be supported without change to the existing interpreter. As discussed previously, the COMPUTE verb could be added if desired, but it would require greater changes both to the grammar and to the interpreter.

Programmers that have used COBOL in a standard implementation will find the appearance of the WORKING-STORAGE SECTION quite different due to the lack of the 77 level. No restriction was placed on the size of the level numbers other than they must be less than 255. This allows for the standard practice of level skipping. In addition, it would not be dificult to make the 77 level perform in a normal manner. There is no difference in the way that the language considers an 01 level and a 77 level item, but the compatability with common usage would be very helpful to a COBOL programmer.

It is hoped that the results of this project are in a form that will allow others to use the compiler as a working system. It is recognized that many undiscovered problems will plague the initial users, but every effort has been made to describe what the system should do and to isolate the functions within the interpreter to facilitate changes.

APPENDIX A - MICRO-COBOL USERS MANUAL

This manual is written to explain the implimentation of HYPO-COBOL done at the Naval Postgraduate School for the Intel 8080 microcomputer running with CP/M (Control Program / Microcomputer). It is not intended that this manual take the place of the HYPO-COBOL specification but that it supply information on the manner in which this implimentation was done. There is no attempt to teach COBOL; however, someone who has a working knowledge of the language should be able to produce programs from the information contained in this manual.

This manual contains a brief overview of the justification for HYPO-COBOL and the organization of this implimentation. It contains a brief explanation of each of the constructs available in the language and shows samples of their use. It explains the interactions between the various parts of the compiler and interpreter and how they interface with the operating system. It also includes a list of references that might be useful to someone who wished to modify the compiler.

One of the major goals of this document is to explain how the operating system used effects the operation of the compiler. It is recognized that if the implimentation is to be useful it will need to be modified to run on other confi-

gurations of hardware and on other operating systems. Where

it was possible, the interaction with the operating environ-

ment was insulated from the other parts of the program, but

in the case of the file structure certain assumptions had to

be made that could require modification.

ACKNOWLEDGEMENT

Any organization interested in reproducing the COBOL
report and specifications in whole or in part, using ideas
from this report as the basis for an instruction manual or
for any other purpose, is free to do so. However, all such
organizations are requested to reproduce the following ack-
nowledgment paragraphs in their entirety as part of the pre-
face to any such publication. Any organization using a
short passage from this document, such as in a book review,
is requested to mention "COBOL" in acknowledgement of the
source, but need not quote the acknowledgement.

> COBOL is an industry language and is not the
> property of any company or group of companies,
> or of any organization or group of organiza-
> tions. No warranty, expressed or implied, is
> made by any contributor or by the CODASYL Pro-
> gramming Language Committee as to the accuracy
> and functioning of the programming system and
> language. Moreover, no responsibility is as-
> sumed by any contributor, or by the committee,
> in connection therewith.

The authors and copyright holders of the copyrighted
material used herein

> FLOW-MATIC (trademark of Sperry Rand Corpora-
> tion), programming for the Univac (R) I and II,
> Data Automation Systems copyrighted 1958, 1959,
> by Sperry Rand Corporation; IBM comercial Trans-
> lator Form No. F 28-8013, copyrighted 1959 by
> IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by
> Minneapolis-Honeywell.

have specifically authorized the use of this material in
whole or in part, in the COBOL specifications. Such author-
ization extends to the reproduction and use of COBOL specif-
ications in programming manuals or similar publications.

CONTENTS

I. HYPO-COBOL OVERVIEW

In order to provide a standard COBOL subset that could
be implimented on a small computer system, the Department
of the Navy has defined HYPO-COBOL. This definition is in-
tended to give the minimum subset of the COBOL language that
would be useable as a working product. This subset does not
agree with the lowest level of COBOL as defined by the CO-
DASYL group and in some cases includes only a portion of one
of the COBOL levels as defined in the current standards. It
is defined to include a portion of the NUCLEUS and both
SEQUENTIAL I-O and RELATIVE I-O. A small portion of the DE-
BUG module was included along with some INTERPROGRAM COMMUN-
ICATION instructions.

Where possible, short forms were included rather than
long forms, and if two forms existed for the same instruc-
tion, only one was included. For example, the shortened PIC
is used rather than the full word PICTURE. Also GO is not
followed by the optional word TO. This does allow the de-
finition to be a proper subset of the standard COBOL, but,
at the same time, reduces the impact of the wordiness of
COBOL on a small system.

As an exception to the general rule, PERFORM UNTIL was
included from level 2 of the NUCLEUS in order to provide an
additional control structure to support structured program-

ming techniques. Further information on HYPO-COBOL can be found in reference 6.

II. ORGANIZATION OF THE IMPLIMENTATION

The compiler is designed to run on an 8080 system in an interactive mode through the use of a teletype or console. It requires at least 12k of RAM memory and a mass storage device for reading and writing. The compiler is composed of two parts or passes, each of which reads a portion of the input file. Pass one reads the input program and builds the symbol table. At the end of the DATA DIVISION, pass one is overlayed by pass two which uses the symbol table to produce the code. The output code is written as it is produced to minimize the use of internal storage.

The first program of the interpreter builds the core image of the code and performs such functions as back-stuffing addresses. This first program loads the second program in and relenquishes control to the run time environment. The interpreter is controlled by a large case statement that decodes the instructions and performs the required actions.

As a tool for debugging the compiler a seperate program was created that will read the output code and translate the operations back into the mnemonics that are used in the second pass of the compiler. This "decode" program has been included with the other programs in order that anyone wishing to make changes to the output code or to the actions of

the interpreter can use this tool.

III. MICRO-COBOL ELEMENTS

This section contains a description of each element in the language and shows simple examples of its use. The following conventions are used in explaining the formats: Elements inclosed in broken braces < > are themselves complete entities and are described elsewhere in the manual. Elements inclosed in stacks of braces { } are choices, one of the elements which is be used. Elements inclosed in brackets [ ] are optional. All elements in capital letters are reserved words and must be spelled exactly.

User names are indicated as lower case. These names have been restricted to 12 characters in length. There are no restrictions in the compiler on what characters may be in a user name. Some restrictions do need to be made to assure that they are not taken as literal numbers when used in the DATA DIVISION. For example a record could be defined in the DATA DIVISION with the name 1234, but the command MOVE 1234 TO RECORD1 would result in the movement of the literal number not the data stored. The HYPO-COBOL description requires that each name start with a letter. This restriction was not implemented because it violates common programming practices.

The input to the compiler does not need to conform to standard COBOL format. Freeform input will be accepted as

68

the default condition. If desired, sequence numbers can be
entered in the first six positions of each line. However, a
toggle needs to be set to cause the compiler to ignore those
lines.

ELEMENT:

IDENTIFICATION DIVISION Format

FORMAT:

IDENTIFICATION DIVISION.

PROGRAM-ID. <comment>.

[AUTHOR. <comment>.]

[DATE-WRITTEN. <comment>.]

[SECURITY. <comment>.]

DESCRIPTION:

This division provides information for program iden-
tification for the reader. The order of the lines is
fixed.

EXAMPLES:

IDENTIFICATION DIVISION.

PROGRAM-ID. SAMPLE.

AUTHOR. A S CRAIG.

ELEMENT:

ENVIRONMENT DIVISION Format

FORMAT:

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. <comment> [DEBUGGING MODE].

OBJECT-COMPUTER. <comment>.

[INPUT-OUTPUT SECTION.

FILE-CONTROL.

    <file-control-entry> . . .

[I-O-CONTROL.

    SAME file-name-1 file-name-2 [file-name-3]

        [file-name-4] [file-name-5].]]

DESCRIPTION:

This division determines the external nature of a
file.   In the case of CP/M all of the files used can
be accessed either sequentially or randomly except for
variable length files which are sequential only. The
debugging mode is also set by this section.

<file-control-entry>

ELEMENT:

'

<file-control-entry>

FORMAT:

1.

SELECT file-name

ASSIGN implementor-name

[ORGANIZATION SEQUENTIAL]

[ACCESS SEQUENTIAL].

2.

SELECT file-name

ASSIGN implementor-name

ORGANIZATION RELATIVE

[ACCESS {SEQUENTIAL [RELATIVE data-name]}].
       {RANDOM RELATIVE data-name   }

DESCRIPTION:

The file-control-entry defines the type of file that
the program expects to see. There is no difference on
the diskette, but the type of reads and writes that
are performed will differ. For CP/M the implementor
name needs to conform to the normal specifications.

EXAMPLES:

        SELECT CARDS

            ASSIGN CARD.FIL.

        SELECT RANDOM-FILE

            ASSIGN A.RAN

            ORGANIZATION RELATIVE

            ACCESS RANDOM RELATIVE RAND-FLAG.

EXAMPLES:

ELEMENT:

DATA DIVISION   Format

FORMAT:

DATA DIVISION.

[FILE SECTION.

[FD file-name

[BLOCK integer-1 RECORDS]

[RECORD [integer-2 TO] integer-3]

[LABEL RECORD {STANDARD}]
                {OMITTED }

[VALUE OF implementor-name-1 literal-1

    [implementor-name-2 literal-2] ...].

[<record-description-entry>] ...] ...

[WORKING-STORAGE SECTION.

[<record-description-entry>] ...]

[LINKAGE SECTION.

[<record-description-entry>] ...]

DESCRIPTION:

This is the section that describes how the data is
structured. There are no major differences from stan-
dard COBOL except for the following: 1. Label
records make no sense on the diskette so no entry is

74

required. 2. The VALUE OF clause likewise has no meaning for CP/M. 3. The linkage section has not been implimented.

If a record is given two lengths as in RECORD 12 TO 128, the file is taken to be variable length and can only be accessed in the sequential mode. See the section on files for more information.

ELEMENT:

<comment>

FORMAT:

any string of characters

DESCRIPTION:

A comment is a string of characters. It may include
anything other than a period followed by a blank or a
reserved word, either of which terminate the string.
Comments may be empty if desired, but the terminator
is still required by the program.

EXAMPLES:

this is a comment

anotheroneallruntogether

8080b 16K

<data-description-entry>

ELEMENT:

   <data-description-entry> Format

FORMAT:

   level-number {data-name}
               {FILLER }

   [REDEFINES data-name]

   [PIC character-string]

   [USAGE {COMP   }]
         {DISPLAY}

   [SIGN {LEADING} [SEPARATE]]
         {TRAILING}

   [OCCURS integer]

   [SYNC [LEFT]]
         [RIGHT]

   [VALUE literal].

DESCRIPTION:

   This statement describes the specific attributes of
   the data. Since the 8080 is a byte machine, there was
   no meaning to the SYNC clause, and thus it has not
   been implimented.

EXAMPLES:

```
01 CARD-RECORD.

    02 PART PIC X(5).

    02 NEXT-PART PIC 99V99 USAGE COMP.

    02 FILLER.

        03 NUMB PIC S9(3)V9 SIGN LEADING SEPARATE.

        03 LONG-NUMB 9(15).

        03 STRING REDEFINES LONG-NUMB PIC X(15).

    02 ARRAY PIC 99 OCCURS 100.
```

ELEMENT:

PROCEDURE DIVISION Format

FORMAT:

1.

PROCEDURE DIVISION [USING name1 [name2] ... [name5]].

section-name SECTION.

[paragraph-name. <sentence> [<sentence> ...] ...] ...

2.

PROCEDURE DIVISION [USING name1 [name2] ... [name5]].

paragraph-name. <sentence> [<sentence> ...] ...

DESCRIPTION:

As is indicated, if the program is to contain sec-
tions, then the first paragraph must be in a section.
The USING option is part of the interprogram communi-
cation module and has not been implimented.

ELEMENT:

    <sentence>

FORMAT:

    <imperative-statement>

    <conditional-statement>

    ENTER verb

DESCRIPTION:

    All sentences other than ENTER fall in one of the two
main catigories. ENTER is part of the interprogram
communication module.

<imperative-statement>

ELEMENT:

<imperative-statement>

FORMAT:

The following verbs are always imperatives:

ACCEPT

CALL

CLOSE

DISPLAY

EXIT

GO

MOVE

OPEN

PERFORM

STOP

The following may be imperatives:

arithmetic verbs without the SIZE ERROR statement

and DELETE, WRITE, and REWRITE without the INVALID option.

<conditional-statements>

ELEMENT:

    <conditional-statements>

FORMAT:

    IF

    READ
                    .
    arithmetic verbs with the SIZE ERROR statement

    and DELETE, WRITE, and REWRITE with the INVALID option.

ELEMENT:

    ACCEPT

FORMAT:

    ACCEPT <identifier>

DESCRIPTION:

    This statement reads up to 72 characters from the con-
    sole. The usage of the item must be DISPLAY.

EXAMPLES:

    ACCEPT IMMAGE

    ACCEPT NUM(9)

ELEMENT:

　　ADD

FORMAT:

　　ADD {identifier} [{identifier-1}] TO identifier-2
　　　　 {literal　　 }　 {literal　　 }

　　　 [ROUNDED] [SIZE ERROR <imperative-statement>].

DESCRIPTION:

　　This instruction adds either one or two numbers　to　a

　　third　 with　 the result being placed in the last loca-

　　tion.

EXAMPLES:
　　　.

　　ADD 10 TO NUMB1

　　ADD X Y TO Z ROUNDED.

　　ADD 100 TO NUMBER SIZE ERROR GO ERROR-LOC

ELEMENT:

CALL

FORMAT:

CALL literal [USING name1 [name2] ... [name5]]

DESCRIPTION:

CALL is not implimented.

ELEMENT:

CLOSE

FORMAT:

CLOSE file-name

DESCRIPTION:

Files must be closed if they have been written.   How-
ever, the normal requirement to close an input file
prior to the end of processing does not exist.

EXAMPLES:

CLOSE FILE1
CLOSE RANDFILE

ELEMENT:

DELETE

FORMAT:

DELETE record-name [INVALID <imperative-statement>]

DESCRIPTION:

This statement requires the record name, not the file
name as in the standard form of the statement. Since
there is no deletion mark in CP/M, this would normally
result in the record still being readable. It is,
therefore, filled with zeroes to indicate that it has
been removed.

EXAMPLES:

DELETE RECORD1

ELEMENT:

DISPLAY

FORMAT:

```
DISPLAY {identifier} [{identifier-1}]
        {literal    } {literal      }
```

DESCRIPTION:

This displays the contents of an identifier or displays a literal on the console. Usage must be DISPLAY. The maximum length of the display is 72 positions.

EXAMPLES:

DISPLAY MESSAGE-1

DISPLAY MESSAGE-3 10

DISPLAY 'THIS MUST BE THE END'

ELEMENT:

DIVIDE

FORMAT:

DIVIDE {identifier} into identifier-1 [ROUNDED]
       {literal   }

       [SIZE ERROR <imperative-statement>]

DESCRIPTION:

The result of the division is stored in identifier-1;

any remainder is lost.

EXAMPLES:

DIVIDE NUMB INTO STORE

DIVIDE 25 INTO RESULT

ELEMENT:

ENTER

FORMAT:

ENTER language-name [routine-name]

DESCRIPTION:

This construct is not implimented.

ELEMENT:

EXIT

FORMAT:

EXIT [PROGRAM]

DESCRIPTION:

The EXIT command causes no action by the interpreter but allows for an empty paragraph for the construction of a common return point. The optional PROGRAM statement is not implimented as it is part of the interprogram communication module.

EXAMPLES:

RETURN.
   EXIT.

ELEMENT:

    GO

FORMAT:

    1.

        GO procedure-name

    2.

        GO procedure-1 [procedure-2] ... procedure-20

            DEPENDING identifier

DESCRIPTION:

        The go command causes an unconditional branch to the
        routine specified.   The second form causes a forward
        branch depending on the value of the contents of the
        identifier.   The identifier must be a numeric integer
        value.   There can be no more than 20 procedure names.

EXAMPLES:

        GO READ-CARD.

        GO READ1 READ2 READ3 DEPENDING READ-INDEX.

ELEMENT:

IF

FORMAT:

IF <condition> {imperative } ELSE imperative-2
                {NEXT SENTENCE}

DESCRIPTION:

This is the standard COBOL IF statement. Note that
there is no nesting of IF statements allowed since the
IF statement is a conditional.

EXAMPLES:

IF A GREATER B ADD A TO C ELSE GO ERROR-ONE.

IF A NOT NUMERIC NEXT SENTENCE ELSE MOVE ZERO TO A.

ELEMENT:

MOVE

FORMAT:

```
MOVE {identifier-1} TO identifier-2
     {literal    }
```

DESCRIPTION:

The standard list of allowable moves applies to this action. As a space saving feature of this implimentation, all numeric moves go through the accumulators. This makes numeric moves slower than alpha-numeric moves, and where possible they should be avoided. Any move that involves picture clauses that are exactly the same can be accomplished as an alpha-numeric move if the elements are redefined as alpha-numeric; also all group moves are alpha-numeric.

EXAMPLES:

MOVE SPACE TO PRINT-LINE.

MOVE A(10) TO B(PTR).

ELEMENT:

MULTIPLY

FORMAT:

MULTIPLY {identifier} BY identifier-2 [ROUNDED]
         {literal   }

    [SIZE ERROR <imperative-statement>]

DESCRIPTION:

The multiply routine requires enough space to calcu-
late the result with the full number of decimal digits
prior to moving the result into identifier-2.   This
means that a number with 5 places after the decimal
multiplied by a number with 6 places after the decimal
will generate a number with 11 decimal places which
would overflow if there were more than 7 digits before
the decimal place.

EXAMPLES:

MULTIPLY X BY Y.

MULTIPLY A BY B(7) SIZE ERROR GO OVERFLOW.

ELEMENT:

OPEN

FORMAT:

```
OPEN {INPUT  file-name }
     {OUTPUT file-name}
     {I-O file-name    }
```

DESCRIPTION:

These three types of opens have the exact same effect
on the diskette. However, they do allow for internal
checking of the other file actions.   For example, a
write to a file set open as input will cause a fatal
error.

EXAMPLES:

OPEN INPUT CARDS.

OPEN OUTPUT REPORT-FILE.

ELEMENT:

PERFORM

FORMAT:

1.

PERFORM procedure-name [THRU procedure-name-2]

2.

PERFORM procedure-name [THRU procedure-name-2]

{identifier} TIMES
{integer }

3.

PERFORM procedure-name [THRU procedure-name-2]

UNTIL <condition>

DESCRIPTION:

All three options are supported. branching may be ei-
ther forward or backward, and the procedures called
may have perform statements in them as long as the end
points do not coincide or overlap.

EXAMPLES:

PERFORM OPEN-ROUTINE.

PERFORM TOTALS THRU END-REPORT.

PERFORM SUM 10 TIMES.

PERFORM SKIP-LINE UNTIL PG-CNT GREATER 60.

ELEMENT:

    READ

FORMAT:

    1.

        READ file-name INVALID <imperative-statement>

    2.

        READ file-name END <imperative-statement>

DESCRIPTION:

    The invalid condition is only applicable to files in a
    random mode. All sequential files must have an END
    statement.

EXAMPLES:

    READ CARDS END GO END-OF-FILE.

    READ RANDOM-FILE INVALID MOVE SPACES TO REC-1.

ELEMENT:

REWRITE

FORMAT:

REWRITE file-name [INVALID <imperative>]

DESCRIPTION:

REWRITE is only valid for files that are open in the
I-O mode. The INVALID clause is only valid for random
files. This statement results in the current record
being written back into the place that it was just
read from. Note that this requires a file name not a
record name.

EXAMPLES:

REWRITE CARDS.

REWRITE RAND-1 INVAID PERFORM ERROR-CHECK.

ELEMENT:

    STOP

FORMAT:

    STOP {RUN   }
        {literal}

DESCRIPTION:

This statement ends the running of the interpreter. If a literal is specified, then the literal is displayed on the console prior to termination of the program.

EXAMPLES:

    STOP RUN.

    STOP 1.

    STOP "INVALID FINISH".

ELEMENT:

SUBTRACT

FORMAT:

SUBTRACT {identifier-1} [identifier-2] FROM identifier-3
         {literal-1 } [literal-2]

   [ROUNDED] [SIZE ERROR <imperative-statement>]

DESCRIPTION:

Identifier-3 is decremented by the value of
identifier/literal one, and, if specified,
identifier/literal two. The results are stored back
in identifier-3. Rounding and size error options are
available if desired.

EXAMPLES:

SUBTRACT 10 FROM SUB(12).
SUBTRACT A B FROM C ROUNDED.

ELEMENT:

WRITE

FORMAT:

1.

    WRITE file-name [{BEFORE} ADVANCING {INTEGER}]
                    {AFTER }   {PAGE   }

2.

    WRITE file-name INVALID <imperative-statement>

DESCRIPTION:

There is no printer on the 8080 system here, so the ADVANCING option is not implimented. The INVALID option only applies to random files.

EXAMPLES:

    WRITE OUT-FILE.

    WRITE RAND-FILE INVALID PERFORM ERROR-RECOV.

ELEMENT:

    <condition>

FORMAT:

   RELATIONAL CONDITION:

```
{identifier-1} [NOT] {GREATER} {identifier-2}
{literal-1}           {LESS   } {literal-2   }
                      {EQUAL   }
```

   CLASS CONDITION:

```
identifier [NOT] {NUMERIC    }
                 {ALPHABETIC}
```

DESCRIPTION:

   It is not valid to compare two literals. The class
   condition NUMERIC will allow for a sign if the iden-
   tifier is signed numeric.

EXAMPLES:

   A NOT LESS 10.

   LINE GREATER "C".

   NUMB1 NOT NUMERIC

ELEMENT:

Subscripting

FORMAT:

data-name (subscript)

DESCRIPTION:

Any item defined with an OCCURS many be referenced by
a subscript. The subscript may be a literal integer,
or it may be a data item that has been specified as an
integer. If the subscript is signed, the sign must be
positive at the time of its use.

EXAMPLES:

A(10)

ITEM(SUB)

IV. COMPILER TOGGLES

There are four toggles in the compiler. They are en-
tered on the first line of the program as a dollar sign fol-
lowed by the given letter. In each case the toggle reverses
the default value.

$L -- list the input code on the screen as the program
is compiled. Default is on. Error messages will be diffi-
cult to understand if this toggle is turned off, but if the
interface device is a teletype, it may be desired in certain
situations.

$S -- sequence numbers are in the first six positions
of each record. Default is off.

$P -- list productions as they occur. Default is off.

$T -- list tokens from the scanner. Default is off.

V. RUN TIME CONVENTIONS

This section explains how to run the compiler on the current system. The compiler expects to see a file with a type of CBL as the input file. In general, the input is free form. If the input includes line numbers then the compiler must be notified by setting the appropriate toggle. The compiler is started by typing COBOL <file-name>. Where the file name is the system name of the input file. There is no interaction required to start the second part of the compiler. The output file will have the same file name as the input file, and will be given a file type of CIM. Any previous copies of the file will be erased.

The interpreter is started by typing CBLINT <file-name>. The first program is a loader, and it will display "LOAD FINISHED" to indicate successful completion. The run-time package will be brought in by the build program, and execution should continue without interuption.

VI. FILE INTERACTIONS WITH CP/M

The file structure that is expected by the program im-
poses some restrictions on the system. References 2 and 3
contain detailed information on the facilities of CP/M, and
should be consulted for details. The information that has
been included in this section is intended to explain where
limitations exist and how the program interacts with the
system.

All files in CP/M are on a random access device, and
there is no way for the system to distinguish sequential
files from files created in a random mode. This means that
the various types of reads and writes are all valid to any
file that has fixed length records. The restrictions of the
ASSIGN statement do prevent a file from being open for both
random and sequential actions during one program.

Each logical record is terminated by a carriage return
and a line feed. In the case of variable length records,
this is the only end mark that exists. This convention was
addopted to allow the various programs which are used in
CP/M to work with the files. Files created by the editor,
for example, will generally be variable length files. This
convention does remove the capability of reading variable
length files in a random mode.

All of the physical records are assumed to be 128 bytes in length, and the program supplies buffer space for these records in addition to the logical records. Logical records may be of any desired length.

ERROR MESSAGES

COMPILER FATAL MESSAGES

BR   Bad read -- disk error, no corrective action can be
        taken in the program.

CL   Close error -- unable to close the output file.

MA   Make error -- could not create the output file.

MO   Memory overflow -- the code and constants generated
        will not fit in the alloted memory space.

OP   Open error -- can not open the input file, or no such
        file present.

ST   Symbol table overflow -- symbol table is too large for
        the allocated space.

WR   Write error -- disk error, could not write a code
        record to the disk.

COMPILER WARNINGS

EL   Extra levels -- only 10 levels are allowed.

FT   File type -- the data element used in a read or write
      statement is not a file name.

IA   Invalid access -- the specified options are not an al-
      lowable combination.

ID   Identifier stack overflow -- more than 20 items in a
      GO TO -- DEPENDING statement.

IS   Invalid subscript -- an item was subscripted but it
      was not defined by an OCCURS.

IT   Invalid type -- the field types do not match for this
      statement.

LE   Literal error -- a literal value was assigned to an
      item that is part of a group item previously assigned
      a value.

NF   No file assigned -- there was no SELECT clause for
      this file.

NI   Not implimented -- a production was used that is not
      implimented.

NN   Non-numeric -- an invalid character was found in a
      numeric string.

NP   No production -- no production exists for the cuurrent
        parser configuration; error recovery will automatical-
        ly occur.

NV   Numeric value -- a numeric value was assigned to a
        non-numeric item.

PC   Picture clause -- an invalid character or set of char-
        acters exists in the picture clause.

PF   Paragraph first -- a section header was produced after
        a paragraph header, which is not in a section.

R1   Redefine nesting -- a redefinition was made for an
        item which is part of a redefined item.

R2   Redefine length -- the length of the redefinition item
        was greater than the item that it redefined.

SE   Scanner error -- the scanner was unable to read an
        identifier due to an invalid character.

SG   Sign error -- either a sign was expected and not
        found, or a sign was present when not valid.

SL   Significance loss -- the number assigned as a value is
        larger than the field defined.

TE   Type error -- the type of a subscript index is not in-
        teger numeric.

VE   Value error -- a value statement was assigned to an
        item in the file section.

INTERPRETER FATAL ERRORS

CL   Close error -- the system was unable to close an out-
        put file.

ME   Make error -- the system was unable to make an input
        file on the disk.

NF   No file -- an input file could not be opened.

WI   Write to input -- a write was attempted to an input
        file.

INTERPRETER WARNING MESSAGES

EM   End mark -- a record that was read did not have a car-
        riage return or a line feed in the expected location.

GD   Go to depending -- the value of the depending indica-
        tor was greater than the number of available branch

addresses.

IC   Invalid character -- an invalid character was loaded
into an output field during an edited move. For exam-
ple, a numeric character into an alphabetic-only
field.

SI   Sign Invalid -- the sign is not a "+" or a "-".

LIST OF REFERENCES

1. Craig, A. S. MICRO-COBOL an implementation of Navy Standard HYPO-COBOL for a microprocessor-based computer system, Masters Thesis, Naval Postgraduate School, March 1977.

2. Digital Research, An Introduction to CP/M Features and Facilities, 1976

3. Digital Research, CP/M Interface Guide, 1976.

4. Intel Corporation, 8008 and 8080 PL/M Programming Manual, 1975.

5. Intel Corperation, 8080 Simulator Software Package, 1974.

6. Software Development Division, ADPE Selection Office, Department of the Navy, HYPO-COBOL, April 1975.

7. Strutynski, Kathryn B. Information on the CP/M Interface Simulator, internally distributed technical note.

```
0C001    1          /*        CCBCL COMPILER - PART 1              */
00002    1
00003    1
0C0C4    1    100H:  /*    LCAC POINT */
0C005    1
00006    1          /*     GLCBAL CECLARATIONS AND LITERALS     */
0CQC7    1
0C008    1    CECLARE LIT LITERALLY 'LITERALLY';
00009    1    DECLARE
CC010    1        800S            LIT        '5H', /* ENTRY TO OPERATING SYSTEM */
0C011    1        MAXSMEMCRY      LIT        '3100H',  /* TOP OF USEABLE MEMORY */
0C012    1        INITIAL$POS     LIT        '2C00H',
0C013    1        RCRSLENGTH      LIT        '255',
00014    1        PASS1SLEN       LIT        '46',
0C015    1        BCCT            LIT        '0',
00016    1        CR              LIT        '13',
000 17   1        LF              LIT        '10',
0C018    1        CUOTE           LIT        '22H',
0C019    1        PCUND           LIT        '23H',
CC020    1        TRLE            LIT        '1',
0C021    1        FALSE           LIT        '0',
0C022    1        FGREVER         LIT        'WHILE TRUE';
CC023    1
CC024    1
0C025    1
0C027    1    DECLARE MAXRNO LITERALLY '104',/* MAX READ CCUNT */
0C028    1            MAXLNC LITERALLY '129',/* MAX LOCK COUNT */
0CC29    1            MAXPNC LITERALLY '145',/* MAX PUSH COUNT */
0C030    1            MAXSNO LITERALLY '234',/* MAX STATE CCUNT */
00031    1            STARTS LITERALLY '1';/* START STATE */
0C032    1
00033    1    DECLARE REAC1 CATA(0,57,48,56,32,8,25,59,2,16,17,22,29,53,58,11,32,32,39
C0034    1        ,38,34,44,9,19,32,37,6,33,3,14,15,18,20,32,28,49,32,1,42,38,36,43,1
00035    1        ,1,1,1,1,1,1,1,1,10,1,39,1,1,1,38,40,49,38,39,1,1,38,23,24,55,52,41
00036    1        ,35,46,1,7,50,1,32,1,32,32,45,1,32,1,32,1,32,47,37,4,26,32,54,40,1,1
0C037    1        ,32,5,12,13,21,22,27,1,60,1,23,24,55,30,51);
00038    1    CECLARE LOCK1 CATA(0,9,0,25,0,9,19,0,42,0,42,0,1,0,52,0,41,0,35,0,1,0,47
0C039    1        ,0,4,0,54,0,40,0,35,40,0,0,1,0,32,0,1,0,1,0,11,0,20,0,7,0,32,0,32,0
0C040    1        ,32,0);
0C041    1    DECLARE APPLY1 CATA(0,0,0,0,0,0,9,10,12,14,19,0,0,0,0,0,0,101,0,0,100,0
0C042    1        ,0,0,0,0,0,97,0,27,0,0,0,69,0,91,92,0,0,91,92,0,0,0,0,13,17,0,102
0C043    1        ,103,1C4,0,0,0,0,0,95,0,0,34,0,0,23,30,38,39,0,21,40,52,56,87,93,94
0C044    1        ,0);
0CC45    1    DECLARE READ2 CATA(0,65,57,64,154,26,37,67,21,30,31,33,39,61,66,27,234
0C046    1        ,215,51,45,1C8,109,223,224,233,43,216,217,22,230,229,232,231,223,173
0C047    1        ,172,169,9,226,47,196,195,7,8,11,13,15,2,3,105,14,158,4,50,20,12,18
0C048    1        ,48,171,170,44,49,15,10,46,35,36,0,3,60,53,42,146,16,25,58,106,155
0C049    1        ,148,155,155,95,15C,155,152,155,157,155,56,193,23,208,234,62,52,2C6
CC050    1        ,180,234,24,28,1C7,52,34,38,17,68,164,35,38,63,40,59);
00051    1    DECLARE LOCK2 CATA(0,5,130,6,131,29,29,132,41,133,54,134,135,69,71,136
0C052    1        ,72,137,73,138,139,80,84,140,86,198,88,141,89,142,184,184,184,51,189
0C053    1        ,92,93,197,211,95,143,90,97,176,99,144,145,101,102,200,103,202,104
CC054    1        ,188);
0C055    1    DECLARE APPLY2 CATA(0,0,77,111,112,147,79,114,81,82,83,78,76,117,75,156
0C056    1        ,126,163,162,100,166,165,167,118,168,160,124,179,178,94,121,74,125
0C057    1        ,120,115,187,187,186,98,152,192,191,194,113,163,128,129,127,205,205
0C058    1        ,205,204,115,123,90,122,214,213,221,219,218,222,199,85,220,116,97
0C059    1        ,110,7C,184,209,207,182,182,181);
0C060    1    DECLARE INCEX1 DATA(0,1,2,3,4,5,6,7,8,4,4,24,4,24,4,13,14,24,109,4,15,16
C0061    1        ,16,24,17,18,19,16,20,22,24,25,26,28,29,34,36,37,24,24,16,33,39,40
00062    1        ,42,43,44,45,46,47,48,49,16,50,38,31,16,52,53,54,55,56,57,58,60,61
00063    1        ,62,63,64,8,65,66,69,70,71,72,73,74,75,77,79,81,83,85,87,88,89,9C,92
00064    1        ,93,94,8,8,16,95,97,97,15,103,104,105,109,24,24,24,1,3,5,8,10,12,14
C0065    1        ,16,18,20,22,24,26,28,30,34,36,38,40,42,44,46,48,50,52,185,149,225
0C066    1        ,227,227,19C,151,153,203,159,210,161,175,212,201,177,1,2,3,2,4,4,5,5
0C067    1        ,6,6,12,13,14,14,15,15,16,16,17,19,19,20,20,20,22,22,23,23,24,24,25
0C068    1        ,25,26,26,27,29,29,31,32,52,35,33,35,38,38,33,33,39,39,39,39,35,42
0C069    1        ,42,43,43,44,44,45,45,48,52,52,53,53,54,54,55,55,56,56,56,56,56,56
0C070    1        ,56,56,58,58,58,59,59,61,61,61,61,61,62,67);
0C071    1    DECLARE INCEX2 DATA(0,1,1,1,1,1,1,1,1,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0C072    1        ,1,1,2,2,1,1,2,1,5,2,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0C073    1        ,1,1,1,2,1,1,1,1,5,3,1,1,1,1,1,1,1,2,2,1,1,1,2,1,1,1,1,2,1,1,1,5,5,1
00074    1        ,2,6,6,1,1,1,4,2,1,1,1,2,2,3,2,2,2,2,2,2,2,2,2,1,2,4,2,2,2,2,2,2,2,2
0C075    1        ,2,2,5,6,29,41,54,69,71,72,73,80,84,88,89,96,99,101,3,9,3,0,3,0,3,0
0C076    1        ,0,1,7,8,1,0,6,0,0,1,3,0,1,2,1,0,0,0,0,1,2,0,3,0,1,2,0,1,5,3,0,0,1
00077    1        ,4,0,0,0,1,2,1,2,2,0,2,3,0,3,0,1,0,0,0,1,4,0,0,1,0,0,0,0,1,1,1,1,1
0C078    1        ,1,0,0,0,0,0,0,0,0,0,0,0,0);
CC079    1
CC0B0    1          /* END CF TABLES */
0C081    1    CECLARE
0C082    1        /* JCINT DECLARATIONS
0C083    1        THESE ITEMS ARE DECLARED TOGETHER IN THIS SECTION
0C084    1        IN ORCER TC FACILITATE THEIR BEING SAVED FCR
0C085    1        THE SECCND PART CF THE COMPILER.
0C086    1        */
CC087    1
0C088    1        CUTPUTSFCB      (33) BYTE INITIAL(0,'            ','CIN',0,0,0,0),
0CC89    1        CEBUGGING       BYTE         INITIAL(FALSE),
CC090    1        PRINT$PROC      BYTE         INITIAL(FALSE),
0C091    1        PRINTSTCKEN     BYTE         INITIAL(FALSE),
0C092    1        LIST$INPLT      BYTE         INITIAL(FALSE),
0C093    1        SEQ$NUM         BYTE         INITIAL (FALSE),
0CC94    1        NEXTSSYM        ADDRESS,
0CC95    1        PCINTER         ACDRESS      INITIAL (100H),
0CC96    1        NEXTSAVAILABLE  ADDRESS      INITIAL (2002H),
0CC97    1        MAXSINTIMEM     ADDRESS      INITIAL (3200H),
0CC98    1        FILESSECSENC    BYTE         INITIAL (FALSE),
0CC99    1        FREESSTCRACE    ACDRESS      INITIAL (2500H),
CC100    1
0C101    1        /* I C BUFFERS AND GLOBALS */
0C1C2    1        INSACCR ADDRESS INITIAL (5CH),
00103    1        INPUTSFCB BASED INSADDR (33) BYTE,
0C104    1        CLTPUTSPTR  ACDRESS,
0C105    1        CUTPLTSBUFF (128) BYTE,
0C106    1        CLTPUTSENC ADDRESS,
0C1C7    1        CUTPUTSCHAR BASEC CUTPUTSPTR BYTE;
CC1C8    1
```

115

```
00109    1
00110    1      MCN1: PROCEDURE (F,A);
00111    1          DECLARE F BYTE, A ADDRESS;
00112    2          GO TO BCCS;
00113    2      END MCN1;
00114    1
00115    1      MCN2: PROCEDURE (F,A) BYTE;
00116    2          DECLARE F BYTE, A ADDRESS;
00117    2          GO TO BCCS;
00118    2      END MCN2;
00119    1
00120    1      PRINTCHAR: PROCEDURE (CHAR);
00121    2          DECLARE CHAR BYTE;
00122    2          CALL MCN1 (2,CHAR);
00123    2      END PRINTCHAR;
00124    1
00125    1      CRLF: PROCEDURE;
00126    2          CALL PRINTCHAR(CR);
00127    2          CALL PRINTCHAR(LF);
00128    2      END CRLF;
00129    1
00130    1      PRINT: PROCEDURE (A);
00131    2          DECLARE A ADDRESS;
00132    2          CALL MCN1 (9,A);
00133    2      END PRINT;
00134    1
00135    1      PRINT$ERROR: PROCEDURE (CODE);
00136    2          DECLARE CODE ADDRESS;
00137    2          CALL CRLF;
00138    2          CALL PRINTCHAR(HIGH(CODE));
00139    2          CALL PRINTCHAR(LOW(CODE));
00140    2      END PRINT$ERROR;
00141    1
00142    1      FATAL$ERROR: PROCEDURE(REASON);
00143    2          DECLARE REASON ADDRESS;
00144    2          CALL PRINT$ERROR(REASON);
00145    2          CALL TIME(1C);
00146    2          GO TO BOOT;
00147    2      END FATAL$ERROR;
00148    1
00149    1      OPEN: PROCEDURE;
00150    2          IF MON2 (15,IN$ADDR)=255 THEN CALL FATAL$ERROR('OP');
00151    2      END OPEN;
00152    1
00153    1      MORE$INPUT: PROCEDURE BYTE;
00154    2          /* READS THE INPUT FILE AND RETURNS TRUE IF A RECORD
00155    2             WAS READ.  FALSE IMPLIES END OF FILE */
00156    2          DECLARE CCNT BYTE;
00157    2          IF (CCNT:=MCN2(20,.INPUT$FCB))>1 THEN CALL FATAL$ERROR('BR');
00158    2          RETURN NOT(CCNT);
00159    2      END MORE$INPUT;
00160    1
00161    1      MAKE: PROCEDURE;
00162    2          /* DELETES ANY EXISTING COPY OF THE OUTPUT FILE
00163    2             AND CREATES A NEW COPY*/
00164    2          CALL MCN1(19,.OUTPUT$FCB);
00165    2          IF MON2(22,.OUTPUT$FCB)=255 THEN CALL FATAL$ERROR('MA');
00166    2      END MAKE;
00167    1
00168    1      WRITE$OUTPUT: PROCEDURE;
00169    2          /* WRITES OUT A BUFFER */
00170    2          CALL MCN1(26,.OUTPUT$BUFF);     /* SET DMA */
00171    2          IF MON2(21,.OUTPUT$FCB)<>0 THEN CALL FATAL$ERROR('WR');
00172    2          CALL MCN1(26,80H);      /* RESET DMA */
00173    2      END WRITE$OUTPUT;
00174    1
00175    1      MOVE: PROCEDURE(SOURCE, DESTINATION, COUNT);
00176    2          /* MOVES FOR THE NUMBER OF BYTES SPECIFIED BY COUNT */
00177    2          DECLARE (SOURCE,DESTINATION) ADDRESS,
00178    2          (S$BYTE BASED SOURCE, D$BYTE BASED DESTINATION, COUNT) BYTE;
00179    2          DO WHILE (COUNT:=COUNT - 1) <> 255;
00180    2              D$BYTE=S$BYTE;
00181    3              SOURCE=SOURCE +1;
00182    3              DESTINATION = DESTINATION + 1;
00183    3          END;
00184    2      END MOVE;
00185    1
00186    1      FILL: PROCEDURE(ADDR,CHAR,COUNT);
00187    2          /* MOVES CHAR INTO ADDR FOR COUNT BYTES */
00188    2          DECLARE ADDR ADDRESS,
00189    2          (CHAR,COUNT,DEST BASED ADDR) BYTE;
00190    2          DO WHILE (COUNT:=COUNT -1)<>255;
00191    2              DEST=CHAR;
00192    3              ADDR=ADDR + 1;
00193    3          END;
00194    2      END FILL;
00195    1
00196    1          /*  *   *   *   *   *   SCANNER LITS *  *   *   *   */
00197    1      DECLARE
00198    1          LITERAL          LIT           '15',
00199    1          INPUT$STR        LIT           '32',
00200    1          PERIOD           LIT           'I',
00201    1          INVALID          LIT           '0';
00202    1
00203    1
00204    1          /*  *  *  *  * SCANNER TABLES *  *  *  *   */
00205    1      DECLARE TOKEN$TABLE DATA
00206    1          /* CONTAINS THE TOKEN NUMBER ONE LESS THAN THE FIRST RESERVED WORD
00207    1          FOR EACH LENGTH OF WORD */
00208    1          (0,0,1,4,5,15,22,32,38,44,47,49,51,55,56,57),
00209    1
00210    1      TABLE DATA('FD','OF','TO','PIC','COMP','DATA','FILE'
00211    1          ,'LEFT','MODE','SAME','SIGN','SYNC','ZERO','BLOCK','LABEL'
00212    1          ,'QUOTE','RIGHT','SPACE','USAGE','VALUE','ACCESS','ASSIGN'
00213    1          ,'AUTHOR','FILLER','OCCURS','RANDOM','RECORD','SELECT'
00214    1          ,'DISPLAY','LEADING','LINKAGE','OMITTED','RECORDS'
00215    1          ,'SECTION','DIVISION','RELATIVE','SECURITY','SEPARATE','STANDARD'
00216    1          ,'TRAILING','DEBUGGING','PROCEDURE','REDEFINES'
00217    1          ,'PROGRAM-ID','SEQUENTIAL','ENVIRONMENT','I-O-CONTROL'
00218    1          ,'DATE-WRITTEN','FILE-CONTROL','INPUT-OUTPUT','ORGANIZATION'
```

116

```
OC219   1           ,'CONFIGURATION','IDENTIFICATION','OBJECT-COMPUTER'
00220   1           ,'SOURCE-COMPUTER','WORKING-STORAGE'),
OC221   1
OC222   1       OFFSET  (16) ADDRESS
OC223   1           /*  NUMBER CF BYTES TO INDEX INTO THE TABLE FOR EACH LENGTH */
00224   1           INITIAL (0,0,0,6,9,45,80,128,170,218,245,265,
00225   1               287,335,348,362),
00226   1
OC227   1       WORD$COUNT DATA
00228   1           /* NUMBER CF WORDS OF EACH SIZE */
00229   1           (0,0,3,1,9,7,8,6,6,3,2,2,4,1,1,3),
OC230   1
OC231   1
OC232   1           MAX$LEN          LIT         '16',
OC233   1           ADD$END          DATA        ('PROCEDURE '),
OC234   1           LOCKED           BYTE        INITIAL (0),
OC235   1           HOLD             BYTE,
00236   1           BUFFER$END       ADDRESS     INITIAL  (100H),
OC237   1           NEXT             BASED       POINTER  BYTE,
OC238   1           INBUFF           LIT         '80H',
OC239   1           CHAR             BYTE,
CC240   1           ACCUM$LENG       LIT         '50',
00241   1           ACCUM            BYTE,
OC242   1           R$ACCUM          (ACCUM$LENG)        BYTE,
OC243   1           DISPLAY          BYTE        INITIAL (0),
OC244   1           DISPLAY$REST     (73)        BYTE,
00245   1           TOKEN            BYTE;               /*RETURNED FROM SCANNER */
OC246   1
OC247   1
OC248   1           /* * * * *   PROCEDURES USED BY THE SCANNER * * * */
OC249   1
CC250   1       NEXT$CHAR: PROCEDURE BYTE;
OC251   2           IF LOCKED THEN
OC252   2           DO;
OC253   2               LOCKED=FALSE;
OC254   3               RETURN (CHAR:=HOLD);
OC255   2           END;
OC256   2           IF (POINTER:=POINTER + 1) >= BUFFER$END THEN
CC257   2           DO;
00258   2               IF NOT MORE$INPUT THEN
OC259   3               DO;
CC260   3                   BUFFER$END=.MEMORY;
00261   4                   POINTER=.ADD$END;
OC262   4               END;
OC263   3               ELSE POINTER=INBUFF;
00264   3           END;
00265   2           RETURN (CHAR:=NEXT);
OC266   2       END NEXT$CHAR;
OC267   1
OC268   1       GET$CHAR: PROCEDURE;
OC269   2           /* THIS PROCEDURE IS CALLED WHEN A NEW CHAR IS NEEDED WITHOUT
OC270   2           THE DIRECT RETURN OF THE CHARACTER*/
OC271   2           CHAR=NEXT$CHAR;
OC272   2       END GET$CHAR;
OC273   1
OC274   1       DISPLAY$LINE: PROCEDURE;
OC275   2           IF NOT LIST$INPUT THEN RETURN;
OC276   2           DISPLAY(DISPLAY + 1) = '$';
OC277   2           CALL PRINT(.DISPLAY$REST);
OC278   2           DISPLAY=0;
OC279   2       END DISPLAY$LINE;
OC280   1
OC281   1
OC282   1       LOAD$DISPLAY: PROCEDURE;
OC283   2           IF DISPLAY < 72 THEN
00284   2               DISPLAY(DISPLAY:=DISPLAY + 1) = CHAR;
OC285   2           CALL GET$CHAR;
OC286   2       END LOAD$DISPLAY;
OC287   1
OC288   1       PUT: PROCEDURE;
OC289   2           IF ACCUM < ACCUM$LENG THEN
CC290   2           ACCUM(ACCUM:=ACCUM+1)=CHAR;
00291   2           CALL LOAD$DISPLAY;
OC292   2       END PUT;
OC293   1
OC294   1       EAT$LINE: PROCEDURE;
OC295   2           DO WHILE CHAR<>CR;
OC296   2               CALL LOAD$DISPLAY;
OC297   3           END;
OC298   2       END EAT$LINE;
CC299   1
CC300   1       GET$NC$BLANK: PROCEDURE;
00301   2           DECLARE (N,I) BYTE;
00302   2           DO FOREVER;
00303   3               IF CHAR = ' ' THEN CALL LOAD$DISPLAY;
OC304   3               ELSE
00305   3               IF CHAR=CR THEN
00306   3               DO;
OC307   3                   CALL DISPLAY$LINE;
OC308   4                   IF SEQ$NUM THEN N=8; ELSE N=2;
OC309   4                   DO I = 1 TO N;
OC310   4                       CALL LOAD$DISPLAY;
00311   5                   END;
00312   4                   IF CHAR = '*' THEN CALL EAT$LINE;
OC313   4                   ELSE
OC314   4                   IF CHAR = ':' THEN
OC315   4                   DO;
00316   4                       IF NOT DEBUGGING THEN CALL EAT$LINE;
OC317   5                       ELSE CALL LOAD$DISPLAY;
OC318   5                   END;
OC319   4               END;
OC320   3               ELSE
OC321   3               RETURN;
OC322   2           END;    /* END OF DO FOREVER */
00323   2       END GET$NC$BLANK;
OC324   1
OC325   1       SPACE: PROCEDURE BYTE;
00326   2           RETURN (CHAR=' ') OR (CHAR=CR);
OC327   2       END SPACE;
OC328   1
```

117

```
00329    1    DELIMITER: PROCEDURE BYTE;
00330    2        /* CHECKS FOR A PERIOD FOLLOWED BY A SPACE OR CR*/
00331    2        IF CHAR <> '.' THEN RETURN FALSE;
00332    2        HOLD=NEXT$CHAR;
00333    2        LOOKED=TRUE;
00334    2        IF SPACE THEN
00335    2        DO;
00336    2            CHAR = '.';
00337    2            RETURN TRUE;
00338    3        END;
00339    2        CHAR='.';
00340    2        RETURN FALSE;
00341    2    END DELIMITER;
00342    1
00343    1    END$OF$TOKEN: PROCEDURE BYTE;
00344    2        RETURN SPACE OR  DELIMITER;
00345    2    END END$OF$TOKEN;
00346    1
00347    1    GET$LITERAL: PROCEDURE BYTE;
00348    2        CALL LOAD$DISPLAY;
00349    2        DO FOREVER;
00350    2            IF CHAR= QUOTE THEN
00351    3            DO;
00352    3                CALL LOAD$DISPLAY;
00353    4                RETURN LITERAL;
00354    4            END;
00355    3            CALL PUT;
00356    3        END;
00357    2    END GET$LITERAL;
00358    1
00359    1
00360    1    LOOK$UP: PROCEDURE BYTE;
00361    2        DECLARE POINT ADDRESS,
00362    2        (HERE BASED POINT,I) BYTE;
00363    2
00364    2        MATCH: PROCEDURE BYTE;
00365    3            DECLARE J BYTE;
00366    3            DO J=I TO ACCUM;
00367    3                IF HERE(J - 1) <> ACCUM(J) THEN RETURN FALSE;
00368    4            END;
00369    3            RETURN TRUE;
00370    3        END MATCH;
00371    2
00372    2        POINT=OFFSET(ACCUM)+ .TABLE;
00373    2        DO I=1 TO WORD$COUNT(ACCUM);
00374    2            IF MATCH THEN RETURN I;
00375    3            POINT = POINT + ACCUM;
00376    3        . END;
00377    2        RETURN FALSE;
00378    2    END LOOK$UP;
00379    1
00380    1    RESERVED$WORD: PROCEDURE BYTE;
00381    2        /* RETURNS THE TOKEN NUMBER OF A RESERVED WORD IF THE CONTENTS OF
00382    2        THE ACCUMULATOR IS A RESERVED WORD, OTHERWISE RETURNS ZERO */
00383    2        DECLARE VALUE BYTE;
00384    2        DECLARE NUMB BYTE;
00385    2            IF ACCUM > MAX$LEN THEN RETURN 0;
00386    2            IF (NUMB:=TOKEN$TABLE(ACCUM))=0 THEN RETURN 0;
00387    2            IF (VALUE:=LOOK$UP)=0 THEN RETURN 0;
00388    2            RETURN (NUMB + VALUE);
00389    2    END RESERVED$WORD;
00390    1
00391    1    GET$TOKEN: PROCEDURE BYTE;
00392    2        ACCUM=0;
00393    2        CALL GET$NONBLANK;
00394    2        IF CHAR=QUOTE THEN RETURN GET$LITERAL;
00395    2        IF DELIMITER THEN
00396    2        . DO;
00397    2            CALL PUT;
00398    3            RETURN PERIOD;
00399    3        END;
00400    2        DO FOREVER;
00401    2            CALL PUT;
00402    3            IF END$OF$TOKEN THEN RETURN INPUT$STR;
00403    3        END; /* OF DO FOREVER */
00404    2    END GET$TOKEN;
00405    1
00406    1
00407    1
00408    1    SCANNER: PROCEDURE;
00409    2        DECLARE CHECK BYTE;
00410    2        DO FOREVER;
00411    2            IF(TOKEN:=GET$TOKEN) = INPUT$STR THEN
00412    2                IF (CHECK:=RESERVED$WORD) <> 0 THEN TOKEN=CHECK;
00413    3            IF TOKEN <> 0 THEN RETURN;
00414    3            CALL PRINT$ERROR ('SE');
00415    3            DO WHILE NOT END$OF$TOKEN;
00416    3                CALL GET$CHAR;
00417    4            END;
00418    3        END;
00419    2    END SCANNER;
00420    1
00421    1
00422    1    PRINT$ACCUM: PROCEDURE;
00423    2        ACCUM(ACCUM+1)='$';
00424    2        CALL PRINT(.R$ACCUM);
00425    2    END PRINT$ACCUM;
00426    1
00427    1    PRINT$NUMBER: PROCEDURE(NUMB);
00428    2        DECLARE(NUMB,I,CNT,K) BYTE, J DATA(100,10);
00429    2        DO I=0 TO 1;
00430    2            CNT=0;
00431    3            DO WHILE NUMB >= (K:=J(I));
00432    3                NUMB=NUMB - K;
00433    4                CNT=CNT + 1;
00434    4            END;
00435    3            CALL PRINTCHAR('0' + CNT);
00436    3        END;
00437    2        CALL PRINTCHAR('0' + NUMB);
00438    2    END PRINT$NUMBER;
```

118

```
0C439   1
CC440   1
0C441   1
0C442   1       INIT$SCANNER: PROCEDURE;
00443   2           /*     INITIALIZE FOR INPUT - OUTPUT OPERATIONS     */
00444   2           CALL MOVE (.'CBL', IN$ADDR + 9, 3);
00445   2           CALL FILL(IN$ADDR + 12,0,5);
0C446   2           CALL OPEN;
00447   2           CALL MOVE(IN$ADDR,.OUTPUT$FCB,9);
00448   2           OUTPUT$ENO=(OUTPUT$PTR:=.OUTPUT$BUFF - 1) + 12B;
00449  ·2           CALL MAKE;
0C450   2           CALL GET$CHAR;     /* PRIME THE SCANNER */
00451   2           DO WHILE CHAR = '$';
00452   2               IF NEXTCHAR = 'L' THEN LIST$INPUT=NOT LIST$INPUT;
0C453   3               ELSE IF CHAR ='S' THEN SEQ$NUM= NOT SEQ$NUM;
0C454   3               ELSE IF CHAR = 'P' THEN PRINT$PROD = NOT PRINT$PROD;
0C455   3               ELSE IF CHAR = 'T' THEN PRINT$TOKEN = NOT PRINT$TOKEN;
0C456   3               CALL GET$CHAR;
0C457   3               CALL GET$NO$BLANK;
0C458   3           END;
00459   2       END INIT$SCANNER;
0C460   1
0C461   1           /*  *   *   *   END OF SCANNER PROCEDURES   *   *   */
0C462   1
0C463   1
0C464   1           /*  *   *   *   *   SYMBOL TABLE DECLARATIONS * * *   */
0C465   1
0C466   1       DECLARE
0C467   1
0C468   1       CUR$SYM                  ADDRESS,        /*SYMBOL BEING ACCESSED*/
0C469   1       SYMBOL                   BASED CUR$SYM  BYTE,
0C470   1       SYMBOL$ADDR              BASED CUR$SYM  ADDRESS,
00471   1       NEXT$SYM$ENTRY           BASED NEXT$SYM ADDRESS,
0C472   1       HASH$PTR                 ADDRESS,
0C473   1       DISPLACEMENT             LIT            '12',
00474   1       HASH$MASK                LIT            '3FH',
CC475   1       S$TYPE                   LIT            '2',
00476   1       OCCURS                   LIT            '11',
0C477   1       ADDR2                    LIT            '4',
0C478   1       P$LENGTH                 LIT            '3',
0C479   1       S$LENGTH                 LIT            '3',
C0480   1       LEVEL                    LIT            '10',
0C481   1       LOCATION                 LIT            '2',
00482   1       REL$IC                   LIT            '5',
CC483   1       START$NAME               LIT            '11',   /*1 LESS*/
00484   1       MAX$IC$LEN               LIT            '12';
0C485   1
0C486   1           /*  *   *   *   *   TYPE LITERALS * * * * * * * */
0C487   1
0C488   1       DECLARE
CC489   1       SEQUENTIAL       LIT           '1',
C0490   1       RANDOM           LIT           '2',
0C491   1       SEQ$RELATIVE     LIT           '3',
0C492   1       VARIABLE$LENG    LIT           '4',
0C493   1       GROUP            LIT           '6',
CC494   1       COMP             LIT           '21';
0C495   1
0C496   1           /*  *   *   *   SYMBOL TABLE ROUTINES   *   *   *   */
0C497   1
0C498   1       INIT$SYMBOL: PROCEDURE;
00499   2           CALL FILL (FREE$STORAGE,0,130);
00500   2           /* INITIALIZE HASH TABLE AND FIRST COLLISION FIELD */
00501   2           NEXT$SYM=FREE$STORAGE+12B;
0C502   2           NEXT$SYM$ENTRY=0;
0C503   2       END INIT$SYMBOL;
0C504   1
00505   1       GET$P$LENGTH: PROCEDURE BYTE;
00506   2           RETURN SYMBOL(P$LENGTH);
0C507   2       END GET$P$LENGTH;
0C508   1
0C509   1       SET$ADDRESS: PROCEDURE(ADDR);
CC510   2       DECLARE ADDR ADDRESS;
00511   2           SYMBOL$ADDR(LOCATION)=ADDR;
00512   2       END SET$ADDRESS;
0C513   1
0C514   1       GET$ADDRESS: PROCEDURE ADDRESS;
00515   2           RETURN SYMBOL$ADDR(LOCATION);
0C516   2       END GET$ADDRESS;
0C517   1
0051B   1       GET$TYPE: PROCEDURE BYTE;
0C519   2           RETURN SYMBOL(S$TYPE);
CC520   2       END GET$TYPE;
0C521   1
0C522   1       SET$TYPE: PROCEDURE(TYPE);
00523   2           DECLARE TYPE BYTE;
00524   2           SYMBOL(S$TYPE)=TYPE;
0C525   2       END SET$TYPE;
0C526   1
00527   1       OR$TYPE: PROCEDURE(TYPE);
00528   2           DECLARE TYPE BYTE;
C0529   2           SYMBOL(S$TYPE)=TYPE OR GET$TYPE;
CC530   2       END OR$TYPE;
0C531   1
00532   1       GET$LEVEL: PROCEDURE BYTE;
00533   2           RETURN SHR(SYMBOL(LEVEL),4);
0C534   2       END GET$LEVEL;
0C535   1
0C536   1       SET$LEVEL: PROCEDURE (LVL);
00537   2           DECLARE LVL BYTE;
0C538   2           SYMBOL(LEVEL)=SHL(LVL,4) OR SYMBOL(LEVEL);
00539   2       END SET$LEVEL;
C0540   1
0C541   1       GET$DECIMAL: PROCEDURE BYTE;
0C542   2           RETURN SYMBOL(LEVEL) AND OFH;
0C543   2       END GET$DECIMAL;
0C544   1
0C545   1       SET$DECIMAL: PROCEDURE (DEC);
00546   2           DECLARE DEC BYTE;
00547   2           SYMBOL(LEVEL) = DEC OR SYMBOL(LEVEL);
0C54B   2       END SET$DECIMAL;
```

119

```
00549    1       SETSSLENGTH: PROCEDURE(HOWSLONG);
00550    1            DECLARE HOWSLONG ADDRESS;
00551    2            SYMBOLSADDR(SSLENGTH) = HCWSLONG;
00552    2       END SETSSLENGTH;
00553    2
00554    1
00555    1       GETSSLENGTH: PROCEDURE ADDRESS;
00556    1            RETURN SYMBOLSADDR(SSLENGTH);
00557    2       END GETSSLENGTH;
00558    2 .
00559    1
00560    1       SET$ADDR2: PROCEDURE (ADDR);
00561    1            DECLARE ADDR ADDRESS;
00562    2       SYMBOLSADDR(ADDR2)=ADDR;
00563    2       END SETSADDR2;
00564    2
00565    1       GET$ADDR2: PROCEDURE ADDRESS;
00566    1            RETURN SYMBOLSADDR(ADDR2);
00567    2       END GETSADDR2;
00568    2
00569    1       SET$OCCURS: PROCEDURE(OCCUR);
00570    1            DECLARE OCCUR BYTE;
00571    2            SYMBOL(OCCURS)=OCCUR;
00572    2       END SETSOCCURS;
00573    2
00574    1       GET$OCCURS: PROCEDURE BYTE;
00575    1            RETURN SYMBOL (OCCURS);
00576    2       END GETSOCCURS;
00577    2
00578    1            /*  *  *  *  PARSER DECLARATIONS  *  *  *  */
00579    1       DECLARE
00580    1       INT                  LIT       '63',     /* CODE FOR INITIALIZE */
00581    1       SCD                  LIT       '66',     /* CODE FOR SET CODE START */
00582    1       PSTACKSIZE           LIT       '30',     /* SIZE CF PARSE STACKS*/
00583    1       STATESTACK           (PSTACKSIZE) BYTE,  /* SAVED STATES */
00584    1       VALUE                (PSTACKSIZE) ADDRESS,    /* TEMP VALUES */
00585    1       VARC                 (51)      BYTE,     /*TEMP CHAR STORE*/
00586    1       IDSSTACK             (10)      ADDRESS   INITIAL (0),
00587    1       IDSSTACKPTR          BYTE      INITIAL(0),
00588    1       HOLDSLIT             BYTE,
00589    1       RESTSHOLDSLIT        (ACCUMSLENG) BYTE,
00590    1       HOLDSSYM             ADDRESS,
00591    1       PENDINGSLITERAL      BYTE INITIAL(FALSE),
00592    1       PENDINGSLITSID       ADDRESS,
00593    1       REDEF                BYTE      INITIAL (FALSE),
00594    1       REDEFSONE            ADDRESS,
00595    1       REDEFSTWO            ADDRESS,
00596    1       TEMPSHOLD            ADDRESS,
00597    1       TEMPSTWO             ADDRESS,
00598    1       COMPILING            BYTE      INITIAL(TRUE),
00599    1       SP                   BYTE      INITIAL (255),
00600    1       MP                   BYTE,
00601    1       MPP1                 BYTE,
00602    1       NOLOCK               BYTE      INITIAL(TRUE),
00603    1       (I,J,K)              BYTE,     /*INDICIES FOR THE PARSER*/
00604    1       STATE                BYTE      INITIAL(STARTS);
00605    1
00606    1            /*  *  *  *  PARSER ROUTINES  *  *  *  =  */
00607    1
00608    1       BYTESOUT: PROCEDURE(ONESBYTE);
00609    2            /* THIS PROCEDURE WRITES ONE BYTE OF OUTPUT ONTO THE DISK
00610    2            IF REQUIRED THE OUTPUT BUFFER IS DUMPED TO THE DISK */
00611    2            DECLARE ONESBYTE BYTE;
00612    2            IF (OUTPUTSPTR:=OUTPUTSPTR + 1)> OUTPUTSEND THEN
00613    2            DO;
00614    3                 CALL WRITESOUTPUT;
00615    3                 OUTPUTSPTR=.OUTPUTSBUFF;
00616    3            END;
00617    2            OUTPUTSCHAR=ONESBYTE;
00618    2       END BYTESOUT;
00619    1
00620    1       STRINGSOUT: PROCEDURE (ADDR,COUNT);
00621    2            DECLARE (ADDR,I,COUNT) ADDRESS, (CHAR BASED ADDR) BYTE;
00622    2            DO I=1 TO COUNT;
00623    2                 CALL BYTESOUT(CHAR);
00624    3                 ADDR=ADDR+1;
00625    3            END;
00626    2       END STRINGSOUT;
00627    1
00628    1       ADDRSOUT: PROCEDURE(ADDR);
00629    2            DECLARE ADDR ADDRESS;
00630    2            CALL BYTESOUT(LOW(ADDR));
00631    2            CALL BYTESOUT(HIGH(ADDR));
00632    2       END ADDRSOUT;
00633    1
00634    1       FILLSSTRING: PROCEDURE(COUNT,CHAR);
00635    2            DECLARE (I,COUNT) ADDRESS, CHAR BYTE;
00636    2            DO I=1 TO COUNT;
00637    2                 CALL BYTESOUT(CHAR);
00638    3            END;
00639    2       END FILLSSTRING;
00640    1
00641    1       STARTSINITIALIZE: PROCEDURE(ADDR,CNT);
00642    2            DECLARE (ADDR,CNT) ADDRESS;
00643    2            CALL BYTESOUT(INT);
00644    2            CALL ADDRSOUT(ADDR);
00645    2            CALL ADDRSOUT(CNT);
00646    2       END STARTSINITIALIZE;
00647    1
00648    1       BUILDSSYMBOL: PROCEDURE(LEN);
00649    2            DECLARE LEN BYTE, TEMP ADDRESS;
00650    2            TEMP=NEXTSSYM;
00651    2            IF (NEXTSSYM:=.SYMBOL(LEN:=LEN+DISPLACEMENT))
00652    2                 > MAXSMEMORY THEN CALL FATALSERROR('ST');
00653    2            CALL FILL (TEMP,C,LEN);
00654    2       END BUILDSSYMBOL;
```

120

```
0C655    1      MATCH: PROCEDURE ADDRESS;
0C656    1          /* CHECKS AN IDENTIFIER TO SEE IF IT IS IN THE SYMBOL
0C657    2          TABLE.  IF IT IS PRESENT, CUR$SYM IS SET FCR ACCESS.
0C658    2          CTHERWISE A NEW ENTRY IS MADE AND THE PRINT NAME
0C659    2          IS ENTERED.  ALL NAMES ARE TRUNCATED TO MAX$ID$LEN*/
0C660    2          DECLARE (POINT,COLLISION BASED POINT) ADDRESS,
0C661    2          (HCLO,I) BYTE;
0C662    2          IF VARC>MAX$ID$LEN
0C663    2              THEN VARC = MAX$ID$LEN;
0C664    2              /* TRUNCATE IF REQUIRED */
0C665    2          HOLD = C;
0C666    2          CC I=1 TO VARC;      /* CALCULATE HASH CODE */
0C667    2              HOLD=HCLO + VARC(I);
0C668    2          ENO;
0C669    3          PCINT=FREE$STORAGE + SHL((HOLD AND HASH$MASK),1);
0C670    2          CC FOREVER;
0C671    2              IF COLLISION=0 THEN
0C672    2                  OC;
0C673    3                      CUR$SYM,COLLISION=NEXT$SYM;
0C674    3                      CALL BUILD$SYMBOL(VARC);
0C675    4                      /* LOAD PRINT NAME */
0C676    4                      SYMBOL(P$LENGTH)=VARC;
0C677    4                      CO I = 1 TO VARC;
0C678    4                          SYMBOL(START$NAME + I)=VARC(I);
0C679    4                      ENO;
0C680    5                      RETURN CUR$SYM;
0C681    4                  END;
0C682    4                  ELSE
0C683    3                  DC;
0C684    3                      CUR$SYM=COLLISION;
0C685    3                      IF (HOLD:=GET$P$LENGTH)=VARC THEN
0C686    4                      CC;
0C687    4                          I=1;
0C688    4                          DO WHILE SYMBOL(START$NAME + I)= VARC(I);
0C689    5                          IF (I:=I+1)>HOLD THEN RETURN (CUR$SYM:=COLLISION);
0C690    5                          ENO;
0C691    6                      ENC;
0C692    5                  ENC;
0C693    4                  POINT=COLLISION;
0C694    3              END;
0C695    3      ENO MATCH;
0C696    2
0C697    1      ALLCCATE: PROCEDURE(BYTES$REQ) ADDRESS;
0C698    1          /* THIS ROUTINE CONTROLS THE ALLOCATION CF SPACE
0C699    2          IN THE MEMCRY OF THE INTERPRETER.  */
0C700    2
0C701    2          DECLARE (HOLD,BYTES$REQ) ADDRESS;
0C702    2          HCLO=NEXT$AVAILABLE;
0C703    2          IF (NEXT$AVAILABLE:=NEXT$AVAILABLE + BYTES$REQ)>MAX$INT$MEM
0C704    2              THEN CALL FATAL$ERROR('MU');
0C705    2          RETURN HCLO;
0C706    2      ENO ALLCCATE;
0C7C7    2
0C7C8    1      SET$REDEF: PROCEDURE(OLD,NEW);
0C709    1          DECLARE (OLD,NEW) ADDRESS;
0C710    2          IF (REDEF:=NOT REDEF) THEN
0C711    2          OC;
0C712    2              REDEF$CNE=OLD;
0C713    2              REDEF$TWO=NEW;
0C714    3          END;
0C715    3          ELSE CALL PRINT$ERROR('R1');
0C716    2      END SET$REDEF;
0C717    2
0C718    1      SET$CUR$SYM: PROCEDURE;
0C719    1          CUR$SYM=ID$STACK(ID$STACK$PTR);
0C720    2      ENO SET$CUR$SYM;
0C721    2
0C722    1      STACK$LEVEL: PROCEDURE BYTE;
0C723    1          CALL SET$CUR$SYM;
0C724    2          RETURN CET$LEVEL;
0C725    2      ENO STACK$LEVEL;
0C726    2
0C727    1      LCAD$LEVEL: PROCEDURE;
0C728    1          DECLARE HOLD ADDRESS;
0C729    2
0C730    2          LCAD$REDEF$ADOR: PROCEDURE;
0C731    2              CLR$SYM=REDEF$CNE;
0C732    3              HCLO=GET$ADDRESS;
0C733    3          END LCAD$REDEF$ADOR;
0C734    3
0C735    2          IF ID$STACK<>0 THEN
0C736    2          DC;
0C737    2              IF VALLE(SP-2)=0 THEN
0C738    2              DC;
0C739    3                  CALL SET$CUR$SYM;
0C740    3                  HCLO=GET$S$LENGTH + GET$ADDRESS;
0C741    4              ENC;
0C742    4              ELSE CALL LCAD$REDEF$ADOR;
0C743    3              IF (ID$STACK$PTR:=ID$STACK$PTR+1)>9 THEN
0C744    3              DC;
0C745    3                  CALL PRINT$ERROR('EL');
0C746    3                  ID$STACK$PTR=9;
0C747    4              ENC;
0C748    4          ENO;
0C749    3          ELSE HCLO=NEXT$AVAILABLE;
0C750    2          ID$STACK(ID$STACK$PTR)=VALUE(MPP1);
0C751    2          CALL SET$CUR$SYM;
0C752    2          CALL SET$ACCRESS(HOLD);
0C753    2      END LCAD$LEVEL;
0C754    2
0C755    1
```

121

```
0C756    1        REDEF$UR$VALUE: PROCEDURE;
00757    2            DECLARE HOLD ADDRESS,
0C758    2                (CEC,K,J,SIGN) BYTE;
0C759    2            IF REDEF THEN
0C760    2            DO;
0C761    2                IF REDEF$TWO=CURSSYM THEN
0C762    3                DO;
0C763    3                    HOLD=GET$$$LENGTH;
00764    4                    CUR$SYM=REDEF$ONE;
00765    4                    IF HOLD>GET$$$LENGTH THEN
0C766    4                    DO;
00767    4                        CALL PRINT$ERROR('R2');
C0768    5                        HOLD=GET$$$LENGTH;
0C769    5                        CURSSYM=REDEF$ONE;
0C770    5                        CALL SET$$$LENGTH(HOLD);
0C771    5                    END;
0C772    4                    REDEF=FALSE;
0C773    4                END;
0C774    3                END;
00775    2            ELSE IF PENDING$LITERAL=0 THEN RETURN;
0C776    2            IF PENDING$LIT$IC<>ID$STACK$PTR THEN RETURN;
0C777    2            CALL START$INITIALIZE(GET$ADDRESS,HOLD:=GET$$$LENGTH);
0C778    2            IF PENDING$LITERAL>2 THEN
0C779    2            DO;
CC780    2                IF PENDING$LITERAL=3 THEN CHAR='0';
0C781    3                ELSE IF PENDING$LITERAL=4 THEN CHAR=' ';
0C782    3                ELSE CHAR=CUOTE;
0C783    3                CALL FILL$STRING(HOLD,CHAR);
0C784    3            END;
0C785    2            ELSE IF PENDING$LITERAL = 2 THEN
0C786    2            DO;
0C787    2                IF HOLD <= HOLD$LIT THEN
0C788    3                    CALL STRING$OUT(.REST$HOLD$LIT,HOLD);
C0789    3                ELSE DO;
0C790    3                    CALL STRING$OUT(.REST$HOLD$LIT,HOLD$LIT);
0C791    4                    CALL FILL$STRING(HOLD - (HOLD$LIT + 1),' ');
0C792    4                END;
0C793    3            END;
0C794    2            ELSE DO;
0C795    2                /* THE NUMBER HANDELER */
0C796    2                DECLARE (DEC,MINUS$SIGN,I,J,LIT$DEC,N$LENGTH,
00797    3                    NUM$BEFORE,NUM$AFTER, TYPE) BYTE, ZONE LIT '10H';
CC798    3
0C799    3                IF((TYPE:=GET$TYPE)<16) OR (TYPE>20) THEN
0C800    3                    CALL PRINT$ERROR('NV');
0C801    3                N$LENGTH=GET$$$LENGTH;
0C802    3                CEC=GET$DECIMAL;
0C803    3                MINUS$SIGN=FALSE;
0C804    3                IF REST$HOLD$LIT='-' THEN
0C805    3                DO;
-00806   3                    MINUS$SIGN=TRUE;
0C807    4                    J=1;
0C808    4                END;
CC809    3                ELSE IF REST$HOLD$LIT='+' THEN J=1;
0C810    3                ELSE J=0;
CC811    3                LIT$DEC=0;
00812    3                DO I=1 TO HOLD$LIT;
0C813    3                    IF HOLD$LIT(I)='.' THEN LIT$DEC=I;
0C814    4                END;
0C815    3                IF LIT$DEC=0 THEN
CC816    3                DO;
0C817    3                    NUM$BEFORE=REST$HOLD$LIT-J;
0C818    4                    NUM$AFTER=0;
CC819    4                END;
CC820    3                ELSE DO;
0C821    3                    NUM$BEFORE=LIT$DEC -J-1;
0C822    4                    NUM$AFTER=REST$HOLD$LIT - LIT$DEC;
0C823    4                END;
00824    3                IF (I:=N$LENGTH - DEC)<NUM$BEFORE THEN
0C825    3                    CALL PRINT$ERROR('SL');
00826    3                IF I>NUM$BEFORE THEN
CC827    3                DO;
0C828    3                    I=I-NUM$BEFORE;
CC829    4                    IF MINUS$SIGN THEN
CC830    4                    DO;
0C831    4                        I=I-1;
0C832    5                        CALL BYTE$OUT('0' + ZONE);
0C833    5                    END;
00834    4                    CALL FILL$STRING(I,'0');
0C835    4                END;
0C836    3                ELSE IF MINUS$SIGN THEN REST$HOLD$LIT(J)=REST$HOLD$LIT(J)+ZONE;
00837    3                CALL STRING$OUT(.REST$HOLD$LIT + J, NUM$BEFORE);
0C838    3                IF NUM$AFTER > DEC THEN NUM$AFTER = DEC;
00839    3                CALL STRING$OUT(.REST$HOLD$LIT + LIT$DEC, NUM$AFTER);
CC840    3                IF (I:=DEC - NUM$AFTER)<>0 THEN
CC841    3                    CALL FILL$STRING(I,'0');
0C842    3            END;
00843    2            PENDING$LITERAL=0;
0C844    2        END REDEF$CR$VALUE;
00845    1
CC846    1        REDUCE$STACK: PROCEDURE;
0C847    2            DECLARE HOLD$LENGTH ADDRESS;
C0848    2            CALL SET$CLR$SYM;
CC849    2            CALL REDEF$CR$VALUE;
0C850    2            HOLD$LENGTH=GET$$$LENGTH;
0C851    2            IF GET$TYPE > 128 THEN
0C852    2            DO;
0C853    2                HOLD$LENGTH=HOLD$LENGTH * GET$OCCURS;
0C854    3            END;
CC855    2            ID$STACK$PTR=ID$STACK$PTR - 1;
0C856    2            CALL SET$CLR$SYM;
0C857    2            CALL SET$$$LENGTH(GET$$$LENGTH + HOLD$LENGTH);
0C858    2            CALL SET$TYPE(GROUP);
CC859    2        END REDUCE$STACK;
CC860    1
```

122

```
00861   1       END$OF$RECORD: PROCEDURE;
00862   2           DO WHILE IC$STACK$PTR<>0;
00863   2               CALL REDUCE$STACK;
00864   3           END;
00865   2           CALL SET$CLR$SYM;
00866   2           CALL REDEF$CR$VALUE;
00867   2           IC$STACK=0;
00868   2           TEMP$HOLD=ALLOCATE(TEMP$TWO:=GET$S$LENGTH);
00869   2       END END$OF$RECORD;
00870   1
00871   1       CONVERT$INTEGER: PROCEDURE;
00872   2           DECLARE INTEGER ADDRESS;
00873   2           INTEGER=0;
00874   2           DO I = 1 TO VARC;
00875   2               INTEGER=SHL(INTEGER,3)+SHL(INTEGER,1)+(VARC(I)-'0');
00876   3           END;
00877   2           VALUE(SP)=INTEGER;
00878   2       END CONVERT$INTEGER;
00879   1
00880   1       CR$VALUE: PROCEDURE(PTR,ATTRIB);
00881   2           DECLARE PTR BYTE, ATTRIB ADDRESS;
00882   2           VALUE(PTR)=VALUE(PTR) OR ATTRIB;
00883   2       END CR$VALUE;
00884   1
00885   1       BUILD$FCB: PROCEDURE;
00886   2           DECLARE TEMP ADDRESS;
00887   2           .DECLARE BUFFER(11) BYTE, (CHAR, I, J) BYTE;
00888   2           CALL FILL(.BUFFER,' ',11);
00889   2           J,I=0;
00890   2           DO WHILE (J < 11) AND (I< VARC);
00891   2               IF (CHAR:=VARC(I:=I+1))='.' THEN J=8;
00892   3               ELSE DO;
00893   3                   BUFFER(J)=CHAR;
00894   4                   J=J+1;
00895   4               END;
00896   3           END;
00897   2           CALL SET$ADDR2(TEMP:=ALLOCATE(164));
00898   2           CALL START$INITIALIZE(TEMP,16);
00899   2           CALL BYTE$CUT(0);
00900   2           CALL STRING$CUT(.BUFFER,11);
00901   2           CALL FILL$STRING(4,0);
00902   2           CALL CR$VALUE(SP-1,1);
00903   2       END BUILD$FCB;
00904   1
00905   1       SET$SIGN: PROCEDURE(NUMB);
00906   2           DECLARE NUMB BYTE;
00907   2           IF GET$TYPE=17 THEN CALL SET$TYPE(VALUE(SP) + NUMB);
00908   2           ELSE CALL PRINT$ERROR('SG');
00909   2           IF VALUE(SP)<>0 THEN CALL SET$S$LENGTH(GET$S$LENGTH + 1);
00910   2       END SET$SIGN;
00911   1
00912   1       PIC$ANALIZER: PROCEDURE;
00913   2           DECLARE     /* WORK AREAS AND VARIABLES */
00914   2           FLAG        BYTE,
00915   2           FIRST       BYTE,
00916   2           COUNT       ADDRESS,
00917   2           BUFFER (31) BYTE,
00918   2           SAVE        BYTE,
00919   2           REPITITIONS ADDRESS,
00920   2           J           BYTE,
00921   2           DEC$COUNT   BYTE,
00922   2           CHAR        BYTE,
00923   2           I           BYTE,
00924   2           TEMP        ADDRESS,
00925   2           TYPE        BYTE,
00926   2
00927   2           /* * * MASKS * * */
00928   2           ALPHA    LIT '0',
00929   2           A$EDIT   LIT '2',
00930   2           A$N      LIT '4',
00931   2           EDIT     LIT '8',
00932   2           NUM      LIT '16',
00933   2           NUM$EDIT LIT '32',
00934   2           DEC      LIT '64',
00935   2           SIGN     LIT '128',
00936   2
00937   2           NUM$MASK         LIT        '10101011B',
00938   2           NUM$ED$MASK      LIT        '100001011B',
00939   2           S$NUM$MASK       LIT        '001011111B',
00940   2           A$E$MASK         LIT        '111111100B',
00941   2           A$N$MASK         LIT        '111010101B',
00942   2           A$N$E$MASK       LIT        '111000000B';
00943   2
00944   2           /* TYPES */
00945   2           NE$TYPE LIT '80',
00946   2           N$TYPE  LIT '16',
00947   2           SN$TYPE LIT '17',
00948   2           A$TYPE  LIT '8',
00949   2           AE$TYPE LIT '72',
00950   2           AN$TYPE LIT '9',
00951   2           ANE$TYPE LIT '73';
00952   2
00953   2           INC$COUNT: PROCEDURE(SWITCH);
00954   3               DECLARE SWITCH BYTE;
00955   3               FLAG=FLAG OR SWITCH;
00956   3               IF (COUNT:=COUNT + 1) < 31 THEN BUFFER(COUNT) = CHAR;
00957   3           END INC$COUNT;
00958   2
00959   2           CHECK: PROCEDURE (MASK) BYTE;
00960   3               /* THIS ROUTINE CHECKS A MASK AGAINST THE
00961   3               FLAG BYTE AND RETURNS TRUE IO THE FLAG
00962   3               HAC NO BITS IN COMMON WITH THE MASK */
00963   3               DECLARE MASK BYTE;
00964   3               RETURN NOT ( (FLAG AND MASK) <> 0);
00965   3           END CHECK;
00966   2
```

123

```
00967    2            PICSALLCCATE: PRCCEDURE(AMT) ADDRESS;
00968    3                DECLARE AMT ADDRESS;
00969    3                IF (MAXSINTSMEM:=MAXSINTSMEM - AMT) < NEXTSAVAILABLE
00970    3                    THEN CALL FATALSERROR ('MO');
00971    3                RETURN MAXSINTSMEM;
00972    3            END PICSALLCCATE;
00973    2
00974    2            /* PRCCEDURE EXECUTION STARTS HERE */
00975    2
00976    2            CCUNT,FLAG,DECSCCUNT=0;
00977    2            /* CHECK FCR EXCESSIVE LENGTH */
00978    2            IF VARC > 20 THEN
00979    2            DO;
00980    2                CALL PRINTSERRCR('PC');
00981    2                RETURN;
00982    3            END;
00983    2            /* SET FLAG BITS AND COUNT LENGTH */
00984    2            I =1;
00985    2            DO WHILE I<=VARC;
00986    3                IF (CHAR:=VARC(I))='A' THEN CALL INCSCCUNT(ALPHA);
00987    3                ELSE IF CHAR ='B' THEN CALL INCSCOUNT(ASEDIT);
00988    3                ELSE IF CHAR ='9' THEN CALL INCSCOUNT(NUM);
00989    3                ELSE IF CHAR ='X' THEN CALL INCSCCUNT(ASN);
00990    3                ELSE IF (CHAR='S') AND (COUNT=0) THEN
00991    3                    FLAG=FLAG OR SIGN;
00992    3                ELSE IF (CHAR = 'V') AND (DECSCOUNT=0) THEN
00993    3                    DECSCCUNT=CCUNT;
00994    3                ELSE IF(CHAR='/') OR (CHAR='0') THEN CALL INCSCCUNT(EDIT);
00995    3                ELSE IF
00996    3                    (CHAR='Z') OR (CHAR=',') OR (CHAR='*') OR
00997    3                    (CHAR='+') OR (CHAR='-') OR (CHAR='S') THEN
00998    3                    CALL INCSCOUNT(NUMSEDIT);
00999    3                ELSE IF (CHAR='.') AND (DECSCOUNT=0) THEN
01000    3                DO;
01001    3                    CALL INCSCOUNT(NUMSEDIT);
01002    4                    DECSCCLNT=CCUNT;
01003    4                END;
01004    3                ELSE IF ((CHAR='C') AND (VARC(I+1)='R')) OR
01005    3                    ((CHAR='D') AND (VARC(I+1)='B')) THEN
01006    3                DO;
01007    3                    CALL INCSCOUNT(NUMSEDIT);
01008    4                    CHAR=VARC(I:=I+1);
01009    4                    CALL INCSCOUNT(NUMSEDIT);
01010    4                END;
01011    3                ELSE IF (CHAR='(') AND (COUNT<>0) THEN
01012    3                DO;
01013    4                    SAVE=VARC(I-1);
01014    4                    REPITITIONS=0;
01015    4                    DO WHILE(CHAR:=VARC(I:=I+1))<>')';
01016    4                        REPITITIONS=SHL(REPITITIONS,3) +
01017    5                            SHL(REPITITIONS,1) +(CHAR - '0');
01018    4                    END;
01019    4                    CHAR=SAVE;
01020    4                    DO J=1 TO REPITITIONS-1;
01021    4                        CALL INCSCOUNT(0);
01022    5                    END;
01023    4                END;
01024    3                ELSE DO;
01025    3                    CALL PRINTSERROR('PC');
01026    4                    RETURN;
01027    4                END;
01028    3                I=I+1;
01029    3            END; /* END CF DO WHILE I<= VARC */
01030    2            /* AT THIS FCINT THE TYPE CAN BE DETERMINED */
01031    2            IF  NOT CHECK(NUMSEDIT) THEN
01032    2            DO;
01033    2                IF CHECK(NUMSEDSMASK) THEN TYPE=NETYPE;
01034    3            END;
01035    2            ELSE IF CHECK(NUMSMASK) THEN TYPE=NTYPE;
01036    2            ELSE IF CHECK(SNUMSMASK) THEN TYPE=SSNSTYPE;
01037    2            ELSE IF CHECK(NOT(ALPHA)) THEN TYPE=ATYPE;
01038    2            ELSE IF CHECK(ASESMASK) THEN TYPE =AETYPE;
01039    2            ELSE IF CHECK(ASNSMASK) THEN TYPE=ANTYPE;
01040    2            ELSE IF CHECK(ASNSESMASK) THEN TYPE=ANETYPE;
01041    2            IF TYPE=0 THEN CALL PRINTSERROR('PC');
01042    2            ELSE DO;
01043    2                IF REDEF THEN CURSSYM=REDEFSTWO;
01044    3                ELSE CLRSSYM = HOLDSSYM;
01045    3                CALL SETSTYPE(TYPE);
01046    3                CALL SETSSLENGTH(COUNT + GETSSSLENGTH);
01047    3                IF (TYPE AND 64) <> 0 THEN
01048    3                DO;
01049    3                    CALL SETSADDR2(TEMP:=PICSALLOCATE(COUNT));
01050    4                    CALL STARTSINITIALIZE(TEMP,COUNT);
01051    4                    CALL STRINGSOUT(.BUFFER + 1,COUNT);
01052    4                END;
01053    3                IF DECSCOUNT<>0 THEN CALL SETSDECIMAL(COUNT-DECSCCUNT);
01054    3            END;
01055    2            END PICSANALIZER;
01056    1
01057    1        SETSFILESATTRIB: PROCEDURE;
01058    2            DECLARE TEMP ADDPESS, TYPE BYTE;
01059    2            IF CURSSYM<>VALUE(MPP1) THEN
01060    2            DO;
01061    2                TEMP=CLRSSYM;
01062    3                CLRSSYM=VALUE(MPP1);
01063    3                SYMBCLSADDR(RELSID)=TEMP;
01064    3            END;
01065    2            IF NCT (TEMP:=VALUE(SP-1)) THEN CALL PRINTSERROR ('NF');
01066    2            ELSE DO;
01067    2                IF TEMP=1 THEN TYPE=SEQUENTIAL;
01068    3                ELSE IF TEMP=15 THEN TYPE=RANDOM;
01069    3                ELSE IF TEMP=9 THEN TYPE=SEQSRELATIVE;
01070    3                ELSE DO;
01071    3                    CALL PRINTSERROR('IA');
01072    4                    TYPE=1;
01073    4                END;
01074    3            END;
01075    2            CALL SETSTYPE(TYPE);
01076    2        END SETSFILESATTRIB;
```

124

```
01077   1
01078   1      LOAD$LITERAL: PROCEDURE;
01079   2          DECLARE I BYTE;
01080   2          IF PENDING$LITERAL <> 0 THEN CALL PRINT$ERROR ('LE');
01081   2          ELSE DO I = 0 TO VARC;
01082   2              HOLD$LIT(I)=VARC(I);
01083   3          END;
01084   2      END LOAD$LITERAL;
01085   1
01086   1
01087   1      CHECK$FOR$LEVEL: PROCEDURE;
01088   2          DECLARE NEW$LEVEL BYTE;
01089   2          HOLD$SYM,CUR$SYM=VALUE(MP-1);
01090   2          CALL SET$LEVEL(NEW$LEVEL:=VALUE(MP-2));
01091   2          IF NEW$LEVEL=1 THEN
01092   2          DO;
01093   2              IF ID$STACK<>0 THEN
01094   3              DO;
01095   3                  IF NOT FILE$SEC$END THEN
01096   4                  DO;
01097   4                      CALL SET$REDEF(ID$STACK,VALUE(MP-1));
01098   5                      VALUE(MP)=1;    /* SET REDEFINE FLAG */
01099   5                  END;
01100   4                  CALL END$OF$RECORD;
01101   4              END;
01102   3          END;
01103   2          ELSE DO WHILE STACK$LEVEL >= NEW$LEVEL;
01104   2              CALL REDUCE$STACK;
01105   2          END;
01106   2      END CHECK$FOR$LEVEL;
01107   1
01108   1
01109   1      CODE$GEN: PROCEDURE(PRODUCTION);
01110   2          DECLARE PRODUCTION BYTE;
01111   2          IF PRINT$PROD THEN
01112   2          DO;
01113   2              CALL CRLF;
01114   2              CALL PRINTCHAR(POUND);
01115   2              CALL PRINT$NUMBER(PRODUCTION);
01116   2          END;
01117   2
01118   2          DO CASE PRODUCTION;
01119   2
01120   2      /*  P R O D U C T I O N S */
01121   2
01122   2      /* CASE 0 NOT USED                                              */
01123   2          ;
01124   3      /*      1    <PROGRAM> ::= <ID-DIV> <E-DIV> <D-DIV> PROCEDURE    */
01125   3          COMPILING=FALSE;
01126   3      /*      2    <ID-DIV> ::= IDENTIFICATION DIVISION . PROGRAM-ID . */
01127   3      /*                       <COMMENT> . <AUTH> <DATE> <SEC>         */
01128   3          ;    /* NO ACTION REQUIRED */
01129   3      /*      3    <AUTH> ::= AUTHOR . <COMMENT> .                     */
01130   3          ;    /* NO ACTION REQUIRED */
01131   3      /*      4            <EMPTY>                                     */
01132   3          ;    /* NO ACTION REQUIRED */
01133   3      /*      5    <DATE> ::= DATE-WRITTEN . <COMMENT> .               */
01134   3          ;    /* NO ACTION REQUIRED */
01135   3      /*      6            <EMPTY>                                     */
01136   3          ;    /* NO ACTION REQUIRED */
01137   3      /*      7    <SEC> ::= SECURITY . <COMMENT> .                    */
01138   3          ;    /* NO ACTION REQUIRED */
01139   3      /*      8            <EMPTY>                                     */
01140   3          ;    /* NO ACTION REQUIRED */
01141   3      /*      9    <COMMENT> ::= <INPUT>                               */
01142   3          ;    /* NO ACTION REQUIRED */
01143   3      /*     10            <COMMENT> <INPUT>                           */
01144   3          ;    /* NO ACTION REQUIRED */
01145   3      /*     11    <E-DIV> ::= ENVIRONMENT DIVISION . CONFIGURATION    */
01146   3      /*     11                SECTION . <SRC-OBJ> <I-O>               */
01147   3          ;    /* NO ACTION REQUIRED */
01148   3      /*     12    <SRC-OBJ> ::= SOURCE-COMPUTER . <COMMENT> <DEBUG> . */
01149   3      /*     12                  OBJECT-COMPUTER . <COMMENT> .         */
01150   3          ;    /* NO ACTION REQUIRED */
01151   3      /*     13    <DEBUG> ::= DEBUGGING MODE                          */
01152   3          DEBUGGING=TRUE;    /* SETS A SCANNER TOGGLE */
01153   3      /*     14            <EMPTY>                                     */
01154   3          ;    /* NO ACTION REQUIRED */
01155   3      /*     15    <I-O> ::= INPUT-OUTPUT SECTION . FILE-CONTROL .     */
01156   3      /*     15              <FILE-CONTROL-LIST> <IO>                  */
01157   3          ;    /* NO ACTION REQUIRED */
01158   3      /*     16            <EMPTY>                                     */
01159   3          ;    /* NO ACTION REQUIRED */
01160   3      /*     17    <FILE-CONTROL-LIST> ::= <FILE-CONTROL-ENTRY>        */
01161   3          ;    /* NO ACTION REQUIRED */
01162   3      /*     18                          <FILE-CONTROL-LIST>          */
01163   3      /*     18                          <FILE-CONTROL-ENTRY>         */
01164   3          ;    /* NO ACTION REQUIRED */
01165   3      /*     19    <FILE-CONTROL-ENTRY> ::= SELECT <ID> <ATTRIBUTE-LIST> . */
01166   3          CALL SET$FILE$ATTRIB;
01167   3      /*     20    <ATTRIBUTE-LIST> ::= <ONE-ATTRIB>                   */
01168   3          ;    /* NO ACTION REQUIRED */
01169   3      /*     21                        <ATTRIBUTE-LIST> <ONE-ATTRIB>  */
01170   3          VALUE(MP)=VALUE(SP) OR VALUE(MP);
01171   3      /*     22    <ONE-ATTRIB> ::= ORGANIZATION <ORG-TYPE>           */
01172   3          VALUE(MP)=VALUE(SP);
01173   3      /*     23                    ACCESS <ACC-TYPE> <RELATIVE>        */
01174   3          VALUE(MP)=VALUE(MPP1) OR VALUE(SP);
01175   3      /*     24                    ASSIGN <INPUT>                      */
01176   3          CALL BUILD$FCB;
01177   3      /*     25    <ORG-TYPE> ::= SEQUENTIAL                           */
01178   3          ;    /* NO ACTION REQUIRED - DEFAULT */
01179   3      /*     26                  RELATIVE                             */
01180   3          CALL OR$VALUE(SP,4);
01181   3      /*     27    <ACC-TYPE> ::= SEQUENTIAL                           */
01182   3          ;    /* NO ACTION REQUIRED - DEFAULT */
01183   3      /*     28                  RANDOM                               */
01184   3          CALL OR$VALUE(SP,2);
01185   3      /*     29    <RELATIVE> ::= RELATIVE <ID>                        */
01186   3          CALL OR$VALUE(MP,8);
```

125

```
01187  3    /*      30              <EMPTY>                                    */
01188  3    :       /* NO ACTION REQUIRED - DEFAULT */
01189  3    /*      31   <IC> ::= I-0-CONTROL . <SAME-LIST>                    */
C1190  3    :
01191  3    /*      32              <EMPTY>                                    */
01192  3    :
01193  3    /*      33   <SAME-LIST> ::= <SAME-ELEMENT>                        */
01194  3    :
01195  3    /*      34                 <SAME-LIST> <SAME-ELEMENT>              */
01196  3    :
01197  3    /*      35   <SAME-ELEMENT> ::= SAME <ID-STRING> .                 */
01198  3    :
01199  3    /*      36   <IC-STRING> ::= <ID>                                  */
01200  3    :
01201  3    /*      37                 <ID-STRING> <ID>                        */
01202  3    :
01203  3
01204  3    /*      38   <C-DIV> ::= DATA DIVISION . <FILE-SECTION> <WCRK>     */
01205  3    /*      38              <LINK>                                     */
01206  3    :       /* NO ACTION REQUIRED */
01207  3    /*      39   <FILE-SECTION> ::= FILE SECTION . <FILE-LIST>         */
01208       FILE$SEC$ENC = TRUE;
01209  3    /*      40              <EMPTY>                                    */
C1210       FILE$SEC$ENC=TRUE;
01211  3    /*      41   <FILE-LIST> ::= <FILES>                               */
01212  3    :       /* NC ACTION REQUIRED */
01213  3    /*      42              <FILE-LIST> <FILES>                        */
01214  3    :       /* NO ACTION REQUIRED */
01215  3    /*      43   <FILES> ::= FD <ID> <FILE-CONTROL> .                  */
01216  3    /*      43              <RECORD-DESCRIPTION>                       */
01217  3    CC;
01218  3        CALL END$OF$RECORD;
01219  4        CURR$SYM=VALUE(MPP1);
C1220  4        CALL SET$ADDRESS(TEMP$HOLD);
01221  4        CALL SET$S$LENGTH(TEMP$TWO);
01222  4    END;
01223  3    /*      44   <FILE-CONTROL> ::= <FILE-LIST>                        */
01224  3    :       /* NO ACTION REQUIRED */
01225  3    /*      45              <EMPTY>                                    */
01226  3    :       /* NO ACTION REQUIRED */
01227  3    /*      46   <FILE-LIST> ::= <FILE-ELEMENT>                        */
01228  3    :       /* NC ACTION REQUIRED */
01229  3    /*      47              <FILE-LIST> <FILE-ELEMENT>                 */
C1230  3    :       /* NO ACTION REQUIRED */
01231  3    /*      48   <FILE-ELEMENT> ::= BLOCK <INTEGER> RECORDS            */
01232  3    :       /* NO ACTION REQUIRED - FILES NEVER BLOCKED */
01233  3    /*      49              RECORD <REC-COUNT>                         */
01234  3    CALL SET$S$LENGTH(VALUE(SP));
01235  3    /*      50              LABEL RECORDS STANDARD                     */
01236  3    :       /* NC ACTION REQUIRED */
01237  3    /*      51              LABEL RECORDS OMITTED                      */
01238  3    :       /* NC ACTION REQUIRED */
01239  3    /*      52              VALUE OF <ID-STRING>                       */
C1240  3    :       /* NC ACTION REQUIRED */
01241  3    /*      53   <REC-COUNT> ::= <INTEGER>                             */
01242  3    :       /* NO ACTION REQUIRED - VALUE(SP) CORRECT */
01243  3    /*      54              <INTEGER> TO <INTEGER>                     */
01244  3    CC;
01245  3    VALUE(MP)=VALUE(SP); /* VARIABLE LENGTH */
01246  4    CALL SET$TYPE(4);    /* SET TO VARIABLE */
01247  4    END;
01248  3    /*      55   <WCRK> ::= WORKING-STORAGE SECTION .                  */
01249  3    /*      55              <RECORD-DESCRIPTION>                       */
C1250  3    :       /* NC ACTION REQUIRED */
01251  3    /*      56              <EMPTY>                                    */
01252  3    :       /* NC ACTION REQUIRED */
01253  3    /*      57   <LINK> ::= LINKAGE SECTION . <RECORD-DESCRIPTION>     */
01254  3    CALL PRINT$ERROR('NI'); /* INTER PROG COMM */
01255  3    /*      58              <EMPTY>                                    */
01256  3    :       /* NO ACTION REQUIRED */
01257  3    /*      59   <RECORD-DESCRIPTION> ::= <LEVEL-ENTRY>                */
01258  3    :       /* NC ACTION REQUIRED */
01259  3    /*      60              <RECORD-DESCRIPTION>                       */
01260  3    /*      60              <LEVEL-ENTRY>                              */
01261  3    :       /* NO ACTION REQUIRED */
01262  3    /*      61   <LEVEL-ENTRY> ::= <INTEGER> <DATA-ID> <REDEFINES>     */
01263  3    /*      61              <DATA-TYPE> .                              */
01264  3    CC;
01265  3        CALL LOAD$LEVEL;
01266  4        IF PENDING$LITERAL<>0 THEN PENDING$LIT$ID=ID$STACK$PTR;
01267  4    END;
01268  3    /*      62   <DATA-ID> ::= <IC>                                    */
01269  3    :       /* NC ACTION REQUIRED */
01270  3    /*      63              FILLER                                     */
01271  3    CC;
01272  3        CLR$SYM, VALUE(SP)=NEXT$SYM;
01273  4        CALL BUILD$SYMBOL(O);
01274  4    END;
01275  3    /*      64   <REDEFINES> ::= REDEFINES <IO>                        */
01276  3    CC;
01277  3        CALL SET$REDEF(VALUE(SP),VALUE(SP-2));
01278  4        VALUE(MP)=1;     /* SET REDEFINE FLAG ON */
01279  4        CALL CHECK$FOR$LEVEL;
C1280  4    END;
C1281  3    /*      65              <EMPTY>                                    */
01282  3    CALL CHECK$FOR$LEVEL;
01283  3    /*      66   <DATA-TYPE> ::= <PROP-LIST>                           */
01284  3    :       /* NO ACTION REQUIRED */
01285  3    /*      67              <EMPTY>                                    */
01286  3    :       /* NC ACTION REQUIRED */
01287  3    /*      68   <PROP-LIST> ::= <DATA-ELEMENT>                        */
01288  3    :       /* NC ACTION REQUIRED */
01289  3    /*      69              <PROP-LIST> <DATA-ELEMENT>                 */
C1290  3    :       /* NO ACTION REQUIRED */
01291  3    /*      70   <DATA-ELEMENT> ::= PIC <INPUT>                        */
01292  3    CALL PIC$ANALIZER;
01293  3    /*      71              USAGE COMP                                 */
01294  3    CALL SET$TYPE(COMP);
01295  3    /*      72              USAGE DISPLAY                              */
01296  3    :       /* NC ACTION REQUIRED - DEFAULT */
```

126

```
01297  3      /*      73                    SIGN LEADING <SEPARATE>              */
01298  3          CALL SETSSIGN(18);
01299  3      /*      74                    SIGN TRAILING <SEPARATE>             */
C1300  3          CALL SETSSIGN(17);
01301  3      /*      75                        OCCURS <INTEGER>                 */
01302  2          CC;
01303  3              CALL CRSTYPE(128);
01304  4              CALL SETSOCCURS(VALUE(SP));
01305  4          END;
01306  3      /*      76                        SYNC <DIRECTION>                 */
01307  3          ;      /* NO ACTION REQUIRED - BYTE MACHINE */
013C8  3      /*      77                        VALUE <LITERAL>                  */
01309  3          CC;
C1310  3              IF NCT FILESSECSEND THEN
01311  4                  DC;
01312  4                      CALL PRINTSERROR('VE');
01313  5                      FENDINGSLITERAL=0;
01314  5                  ENC;
01315  4          END;
01316  2      /*      78     <CIRECTION> ::= LEFT                                */
01317  3          ;      /* NO ACTICN REQUIRED */
01318  3      /*      79                        RIGHT                            */
01319  3          ;      /* NC ACTICN REQUIRED */
01320  3      /*      80                        <EMPTY>                          */
01321  3          ;      /* NC ACTICN REQUIRED */
01322  3      /*      81     <SEPARATE> ::= SEPARATE                             */
01323  3          VALUE(SF)=2;
01324  3      /*      82                        <EMPTY>                          */
01325  3          ;      /* NO ACTICN REQUIRED */
01326  3      /*      83     <LITERAL> ::= <INPUT>                               */
01327  3          CC;
01328  3              CALL LCADSLITERAL;
01329  4              PENCINGSLITERAL=1;
C1330  4          END;
01331  3      /*      84                        <LIT>                            */
01332  3          CC;
01333  3              CALL LCADSLITERAL;
01334  4              PENCINGSLITERAL=2;
01335  4          END;
01336  3      /*      85                        ZERC                             */
01337  2          PENCINGSLITEPAL=3;
01338  2      /*      86                        SPACE                            */
01339  2          FENDINGSLITEPAL=4;
C1340  3      /*      87                        QUOTE                            */
01341  3          PENDINGSLITEPAL=5;
01342  3      /*      88     <INTEGER> ::= <INPUT>                               */
01343  3          CALL CCNVERTSINTEGER;
01344  3      /*      89     <IC> ::= <INPUT>                                    */
01345  3          VALUE(SP)=MATCH;      /* STORE SYMBOL TABLE FCINTERS */
01346  2
01347  2
01348  3          END;   /* ENC OF CASE STATEMENT */
01349  2      END CCCESGEN;
C1350  1
01351  1      GETINI: PRCCECURE BYTE;
01352  2          RETLRN INCEX1(STATE);
01353  2      END CETIN1;
01354  1
01555  1      GETIN2: PRCCECLRE BYTE;
01356  2          FETURN INCEX2(STATE);
01357  2      END GETIN2;
01358  1
01359  1      INCSF: PROCECURE;
C1360  2          SF=SP + 1;
01361  2          IF SP >= PSTACKSIZE THEN CALL FATALSERROR('SO');
01362  2          VALUE(SF)=C;   /* CLEAR VALUE STACK */
01363  2      END INCSP;
01364  1
01365  1      LCCKAHEAD: FRCCECURE;
01366  2          IF NCLCCK THEN
01367  2          CC;
01368  3              CALL SCANNER;
01369  3              NCLCCK=FALSE;
C1370  3              IF PRINTSTOKEN THEN
01371  3                  CC;
01372  3                  CALL CRLF;
01373  4                  CALL PRINTSNUMBER(TOKEN);
01374  4                  CALL PRINTSCHAR(' ');
01375  4                  CALL PRINTSACCUM;
01376  4              ENC;
01377  3          END;
01378  2      END LCCKAHEAC;
C1379  1
01380  1      NCSCCNFLICT: PRCCEDURE (CSTATE) BYTE;
01381  2          CECLARE (CSTATE,1,J,K) BYTE;
01382  2          J=INCEX1(CSTATE);
01383  2          K=J + INDE)2(CSTATE) - 1;
01384  2          CC I=J TC K;
01385  2              IF REAC1(I)=TCKEN THEN RETURN TRUE;
01386  2          END;
01387  2      RETLRN FALSE;
01388  2      END NCSCCNFLICT;
01389  1
C1390  1      RECCVER: PRCCECLRE BYTE;
01391  2          CECLARE (TSF, RSTATE) BYTE;
01392  2          CC FOREVER;
01393  3              TSF=SF;
01394  3              CO WHILE TSP <> 255;
01395  3                  IF NOSCCNFLICT(RSTATE:=STATESTACK(TSP)) THEN
01396  4                      CC;  /* STATE WILL READ TOKEN */
01397  4                          IF SP<>TSP THEN SP = TSP - 1;
01398  5                          RETURN RSTATE;
01399  5                      END;
014C0  4                      TSP = TSP - 1;
01401  4                  ENC;
01402  3                  CALL SCANNER; /* TRY ANOTHER TOKEN */
01403  3              END;
01404  2      END RECCVER;
```

127

```
01405    1         ENC$PASS: PROCEDURE;
01406    1             /* THIS PROCEDURE STORES THE INFORMATION REQUIRED BY PASS2
01407    2         IN LOCATIONS ABOVE THE SYMBOL TABLE. THE FOLLOWING
01408    2         INFORMATION IS STORED:
01409    2             OUTPUT FILE CONTROL BLOCK
01410    2             COMPILER TOGGLES
01411    2             INPUT BUFFER POINTER
01412    2         THE OUTPUT BUFFER IS ALSO FILLED SO THE CURRENT RECORD IS WRITTEN.
01413    2         */
01414    2
01415    2
01416    2             CALL BYTE$CLT(SCD);
01417    2             CALL ADDR$CLT(NEXT$AVAILABLE);
01418    2             DC WHILE CLTPUT$PTR<>.OUTPUT$BUFF;
01419    2                 CALL BYTE$OUT(OFFH);
01420    3             END;
01421    2
01422    2             CALL MOVE(.OUTPUT$FCB,MAX$MEMORY-PASS1$LEN,PASS1$LEN);
01423    2             GO TO MAX$MEMORY;
01424    2         ENC ENDSPASS;
01425    1
01426    1             /* *   *   *   *   *   PROGRAM EXECUTION STARTS HERE *   *   */
01427    1
01428    1         CALL MOVE(INITIAL$POS,MAX$MEMORY,RDR$LENGTH);
01429    1         CALL INIT$SCANNER;
01430    1         CALL INIT$SYMBOL;
01431    1
01432    1
01433    1             /* *   *   *   *   *   *   PARSER *   *   *   *   *   */
01434    1
01435    1         DC WHILE COMPILING;
01436    1             IF STATE <= MAXRNO THEN             /* READ STATE */
01437    2             CC;
01438    2                 CALL INCSP;
01439    3                 STATESTACK(SP) = STATE;  /* SAVE CURRENT STATE */
01440    3                 CALL LOOKAHEAD;
01441    3                 I=GETIN1;
01442    3                 J = I + GETIN2 - 1;
01443    3                 CC I=I TO J;
01444    3                     IF READ1(I) = TOKEN THEN
01445    4                     CC;
01446    4                     /* COPY THE ACCUMULATOR IF IT IS AN INPUT
01447    4                     STRING.  IF IT IS A RESERVED WORD IT DOES
01448    4                     NOT NEED TO BE COPIED. */
01449    4                         IF (TOKEN=INPUT$STR) OR (TOKEN=LITERAL) THEN
01450    5                             DO K=0 TO ACCUM;
01451    5                                 VARC(K)=ACCUM(K);
01452    6                             END;
01453    5                         STATE=READ2(I);
01454    5                         NOLOOK=TRUE;
01455    5                         I=J;
01456    5                     END;
01457    4                     ELSE
01458    4                     IF I=J THEN
01459    4                     CC;
01460    4                         CALL PRINT$ERROR('NP');
01461    5                         CALL PRINT(.' ERROR NEAR $');
01462    5                         CALL PRINT$ACCUM;
01463    5                         IF (STATE:=RECOVER)=0 THEN COMPILING=FALSE;
01464    5                     END;
01465    4                 ENC;
01466    3             END;    /* ENC OF READ STATE */
01467    2             ELSE
01468    2             IF STATE>MAXRNO THEN         /* APPLY PRODUCTION STATE */
01469    2             CC;
01470    2                 MP=SP - GETIN2;
01471    3                 MPP1=MP + 1;
01472    3                 CALL CODE$GEN(STATE - MAXRNO);
01473    3                 SP=MP;
01474    3                 I=GETIN1;
01475    3                 J=STATESTACK(SP);
01476    3                 DO WHILE (K:=APPLY1(I)) <> 0 AND J<>K;
01477    3                     I=I + 1;
01478    4                 ENC;
01479    3                 IF (K:=APPLY2(I))=0 THEN COMPILING=FALSE;
01480    3                 STATE=K;
01481    3             END;
01482    2             ELSE
01483    2             IF STATE<=MAXLNO THEN         /*LOOKAHEAD STATE*/
01484    2             CC;
01485    2                 I=GETIN1;
01486    3                 CALL LOOKAHEAD;
01487    3                 OC WHILE (K:=LOOK1(I))<>0 AND TOKEN <>K;
01488    3                     I=I+1;
01489    4                 ENC;
01490    3                 STATE=LOOK2(I);
01491    3             END;
01492    2             ELSE
01493    2             CC;               /*PUSH STATES*/
01494    2                 CALL INCSP;
01495    3                 STATESTACK(SP)=GETIN2;
01496    3                 STATE=GETIN1;
01497    3             END;
01498    2         END;  /* CF WHILE COMPILING */
01499    1         CALL CRLF;
01500    1         CALL PRINT(.'ENC CF PART 1   $');
01501    1         CALL ENDSPASS;
01502    1         ECF
```

128

```
00C01   1
C0C02   1               /*        COBCL COMPILER - PART 2             */
C0C03   1
00C04   1
C0C05   1       10CH:  /*    LCAC POINT */
C0C06   1
00007   1               /*    CLCBAL DECLARATIONS AND LITERALS     */
00008   1
C0C09   1       CECLARE LIT LITERALLY 'LITERALLY';
C0C10   1       CECLARE
00011   1           8COS             LIT        '5H', /* ENTRY TO CPERATING SYSTEM */
C0012   1           FASHSTABSACCR    LIT        '2500H', /* ADDRESS OF THE BCTTOM CF
C0C13   1                                          THE TABLES  FROM PART1 */
00014   1           PASS1SLEN        LIT        '46',
00015   1           MAXSMEMCRY       LIT        '3200H',
C0016   1           PASS1STCP        LIT        '3100H',
00017   1           8CCT             LIT        '0',
CCC18   1           CR               LIT        '13',
C0C19   1           LF               LIT        '10',
C0C20   1           CUCTE            LIT        '22H',
00021   1           PCLND            LIT        '23H',
00022   1           TRUE             LIT        '1',
C0C23   1           FALSE            LIT        '0',
C0C24   1           FCREVER          LIT        'WHILE TRUE';
00025   1       CECLARE MAXRNO LITERALLY '83',/* MAX READ COUNT */
C0026   1               MAXLNC LITERALLY '106',/* MAX LOOK CCUNT */
C0C27   1               MAXFNO LITERALLY '121',/* MAX PUSH COUNT */
C0028   1               MAXSNO LITERALLY '218',/* MAX STATE CCLNT */
C0C29   1               STARTS LITERALLY '1';/* START STATE */
C0C30   1       DECLARE READ1 CATA(0,62,5,6,8,13,15,19,21,23,25,30,31,40,41,43,44,48,52
00031   1           ,53,57,59,47,27,28,35,36,47,27,47,1,28,58,1C,34,45,33,12,27,28,35,36
0C032   1           ,47,3,1,39,22,47,56,1,55,2,29,42,26,18,32,49,51,63,17,4,37,27,38,47
00033   1           ,6C,54,1,14,11,7,9,50,5,8,13,15,19,21,23,25,30,40,41,43,44,48,52,53
J0034   1           ,57,59,5C,7,16,1,1,5,8,13,15,19,20,21,23,25,30,40,41,43,44,48,52,53
00035   1           ,57,59,47,61,9,47,24,0,0);
C0C36   1       CECLARE LOCK1 CATA(0,47,0,39,0,2,0,39,0,1,14,C,47,0,29,42,0,2,0,26,C,7,0
C0037   1           ,16,0,1,14,C,54,0,54,0,54,0,54,0,11,0,1,14,C,1,0,50,0,47,0,24,C,9,47
00038   1           ,C);
C0C39   1       DECLARE APPLY1 CATA(0,C,24,0,6,0,0,79,0,0,83,0,13,69,7C,76,81,0,C,3,83,C
C0040   1           ,3,83,0,27,C,C,0,0,59,60,61,0,0,0,0,0,0,0,71,0,0,0,0,C,C,5,8,9,15,16
C0041   1           ,46,C,0,2,5,6,8,9,14,15,16,20,23,25,28,29,30,31,35,36,42,46,77,78
00042   1           ,79,82,0,10,32,39,40,51,54,56,0,5,8,9,15,16,30,46,C,54,0,22,0,C,17
C0C43   1           ,34,66,67,C,C,0,3,83,0);
C0044   1       DECLARE READ2 CATA(0,43,6,7,10,12,84,17,19,20,22,25,26,29,30,31,32,34,35
C0C45   1           ,36,39,40,33,202,206,206,207,86,202,86,122,85,179,195,193,194,186
00046   1           ,173,211,2C6,2C8,207,210,203,130,28,192,198,87,3,37,4,190,189,23,163
C0047   1           ,169,167,16C,163,16,5,182,202,27,86,41,17J,2,13,165,8,175,185,6,10
C0048   1           ,12,84,17,19,20,22,25,29,30,31,32,34,35,36,39,40,185,9,15,131,132,6
C0C49   1           ,10,12,84,17,18,19,20,22,25,29,30,31,32,34,35,36,39,4C,199,42,11,199
C0C50   1           ,21,0,0);
00C51   1       CECLARE LOCK2 CATA(0,14,107,24,108,199,200,38,1C9,143,143,125,46,110,47
C0052   1           ,47,111,48,197,49,112,113,51,114,54,115,115,56,58,116,159,117,6C,118
C0C53   1           ,61,119,64,120,121,121,66,148,69,71,140,77,123,80,137,129,129,83);
00C54   1       DECLARE APPLY2 CATA(0,0,158,62,78,104,79,128,127,106,75,74,152,151,153
C0C55   1           ,178,15C,132,134,105,105,137,103,103,140,183,76,161,50,67,156,154
00C56   1           ,157,155,149,70,135,63,95,147,68,174,81,16C,57,187,82,97,145,98,99
C0057   1           ,96,176,136,191,44,91,88,91,91,216,91,91,218,18C,139,69,125,90,91
C0058   1           ,158,92,159,144,91,126,126,44,146,45,93,52,53,94,2C4,2C4,55,212,196
C0059   1           ,196,196,196,196,196,196,201,73,72,209,213,172,65,101,214,164,100
C0C60   1           ,141,142,1C2,102,148);
00C61   1       DECLARE INDEX1 CATA(0,1,116,2,22,116,116,23,116,116,28,30,31,74,116,116
C0C62   1           ,116,23,32,116,26,37,116,45,116,116,23,116,116,116,116,28,43,23
C0C63   1           ,116,116,44,45,28,28,40,116,48,49,51,116,52,51,54,55,28,60,61,28,62
C0064   1           ,63,66,67,67,67,67,68,69,70,71,23,23,74,72,74,92,93,94,95,96,97,116
C0065   1           ,116,118,12C,74,116,2,1,3,5,7,9,12,14,17,19,21,23,25,28,30,32,34,36
C0066   1           ,38,41,43,45,47,49,217,124,124,177,188,181,2C5,2C5,184,171,171,171
C0067   1           ,171,166,215,1,2,2,4,4,6,6,7,7,9,9,10,10,10,12,12,12,12,12,12,12,12
C0068   1           ,12,12,12,12,12,12,18,18,18,18,19,19,19,19,22,22,22,25,27,27,27,28
C0069   1           ,28,29,29,29,30,30,34,34,35,35,36,36,37,37,38,38,39,39,40,42,43
C0070   1           ,43,44,44,45,45,46,46,46,47,47,54,55,80,80,80,88,96,96,98,98,98,100
C0071   1           ,1CC,1CC,1C1,1C1,1C6,106,1C7,107,103);
C0C72   1       DECLARE INDEX2 CATA(0,1,1,20,1,1,1,5,1,1,2,1,1,18,1,1,1,5,1,3,1,1,6,1,1
C0073   1           ,1,1,5,1,1,1,1,2,1,5,1,1,1,1,2,2,1,1,2,1,1,2,1,1,5,2,1,1,2,1,3,1,1
C0074   1           ,1,1,1,1,1,1,5,1,5,18,2,18,1,1,1,1,1,19,1,2,2,1,13,1,20,2,2,2,2,3,2
C0075   1           ,3,2,2,2,2,3,2,2,2,2,2,3,2,2,2,2,3,14,24,38,46,47,49,51,54,56,58,59
C0C76   1           ,6C,61,64,66,6,1,0,C,1,0,1,2,2,1,2,1,0,2,1,C,2,1,0,2,1,1,3,3,2,2,0,1
00C77   1           ,2,2,4,2,5,4,4,5,1,1,2,2,0,0,0,1,0,0,0,C,0,0,C,0,0,0,1,1,C,1,1
00C78   1           ,C,0,1,2,0,C,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,C,0,0,C,C,0,0,0,0,0,0,0
C0C79   1           ,1);
C0C80   1
J0C81   1               /* END CF TABLES */
0CC82   1       CECLARE
00C83   1               /* JOINT CECLARATICNS */
00084   1               /* THE FCLLCWING ITEMS ARE DECLARED TCGETHER IN THIS
0CC85   1       GPCUP IN ORCER TO FACILITATE THEIR BEING PASSED FRCM
0CC86   1       THE FIRST PART OF THE COMPILER.
0CC87   1               */
CUC88   1           CUTPUT$FCB       (33) BYTE,
CCC89   1           CEBUGGING        BYTE,
CCC90   1           PRINT$PRCD       BYTE,
00C91   1           PRINT$TCKEN      BYTE,
0CC92   1           LIST$INFLT       BYTE,
0CC93   1           SECSNUM          BYTE,
00094   1           NEXT$SYM         ADDRESS,
0CC95   1           PCINTER          ADCRESS,     /* ROINTS TO THE NEXT BYTE TC BE REAC */
C0C96   1           NEXT$AVAILABLE   ADCRESS,
0CC97   1           MAX$INT$MEM      ADDRESS;
CCC98   1
0CC99   1               /* I D EUFFERS AND GLOBALS */
C0100   1           IN$ACCR ADCRESS INITIAL (5CH),
00101   1           INPUTFCB BASED INADDR (33) BYTE,
00102   1           CLTPUT$ELFF      (128)          BYTE,
00103   1           OUTFUT$PTR               ADDRESS,
C0104   1           CLTPUT$END              ADDRESS,
C0105   1           CUTPUT$CFAR     BASED OUTPUT$PTR BYTE;
```

129

```
UU1U6     1
00107     1          /* GLOBAL CCLNTERS */
C0108     1     CECLARE
C0109     1          CTR BYTE,
C0110     1          ASCTR ADDRESS,
00111     1          BASE ADDRESS,
00112     1          BSBYTE EASEC BASE BYTE,
00113     1          BSADDR BASEC BASE ADDRESS;
C0114     1
C0115     1     MCN1: PRCCECLRE (F,A);
C011C     2          CECLARE F BYTE, A ADDRESS;
00117     2          GC TO BCCS;
C0118     2     END MCN1;
C011S     1
C0120     1     MCN2: PROCECLRE (F,A) BYTE;
00121     2          CECLARE F BYTE, A ADDRESS;
00122     2          GC TC BCCS;
00123     2     END MCN2;
00124     1
00125     1     PRINTCHAR: FRCCECURE (CHAR);
00126     2          CECLARE CHAR BYTE;
00127     2          CALL MCN1 (2,CHAR);
00128     2     ENC PRINTCHAR;
C0129     1
CC130     1     CRLF: PRDCECLRE;
00131     2          CALL PRINTCHAR(CR);
00132     2          CALL PRINTCHAR(LF);
00133     2     ENC CRLF;
C0134     1
00135     1     PRINT: PROCECURE (A);
00136     2          CECLARE A ACDRESS;
00137     2          CALL MCN1 (S,A);
0013E     2     ENC PRINT;
C0139     1
C0140     1     PRINTSERDR: PRCCECURE (CODE);
C0141     2          CECLARE CCDE ADDRESS;
00142     2          CALL CRLF;
00143     2          CALL PRINTCHAR(HIGH(CODE));
00144     2          CALL PRINTCHAR(LCW(CCDE));
00145     2     END PRINTSERFCR;
00146     1
00147     1     FATALSERROR: PRCCECURE(REASCN);
00148     2          CECLARE REASCN ADDRESS;
C0149     2          CALL PRINTSERFCR(REASON);
C0150     2          CALL TIME(10);
0C151     2          GC TO BCCT;
C0152     2     ENC FATALSERRCR;
00153     1
00154     1     CLCSE: FRCCECURE;
00155     1          IF MCN2(16,..CUTPUTSFC8)=255 THEN CALL FATALSERRDR('CL');
00156     2     ENC CLCSE;
C0157     1
C015E     1     MCRESINPLT: FRCCECURE BYTE;
C0159     2          /* READS THE INPUT FILE AND RETURNS TRUE IF A RECORD
C0160     2              WAS REAC.  FALSE IMPLIES END OF FILE */
C0161     2          CECLARE CCNT BYTE;
00162     2          IF (CCNT:=MCN2(20,..INPUTSFC8))>1 THEN CALL FATALSERRCR('BR');
00163     2          RETURN NCT(CCNT);
00164     2     ENC MCRESINPLT;
00165     1
00166     1     WRITESCUTPUT: PROCECURE (LCCATICN);
00167     2          /* WRITES CLT A 128 BYTE BUFFER FROM LOCATICN*/
00168     2          CECLARE LCCATICN ADDRESS;
C0169     2          CALL MCN1(26,LCCATICN);  /* SET DMA */
00170     2          IF MCN2(21,..CUTPUTSFCB)<>0 THEN CALL FATALSERROR('WR');
00171     2          CALL MCN1(26,80H);  /*RESET DMA */
00172     2     ENC WRITESCUTPLT;
00173     1
C0174     1     MCVE: PROCECLRE(SCURCE, DESTINATICN, CCUNT);
00175     2          /* MCVES FCR THE NUMBER OF BYTES SPECIFIED BY COUNT */
00176     2          CECLARE (SCURCE,DESTINATION) ADDRESS,
00177     2          (SSBYTE BASEC SOURCE, DSBYTE BASED DESTINATION, COUNT) BYTE;
00178     2          CC WHILE (CCUNT:=CCUNT - 1) <> 255;
00175     2               DSBYTE=SSBYTE;
CC180     3               SGLRCE=SCURCE +1;
00181     3               DESTINATICN = DESTINATION + 1;
CC182     3          ENC;
0C183     2     END MCVE;
C0184     1
0C185     1     FILL: PROCECLRE(ADDR,CHAR,CCUNT);
00186     2          /* MOVES CHAR INTC ADDR FDR CCUNT BYTES */
0C187     2          CECLARE ACDR ADDRESS,
C0188     2          (CHAR,CCLNT,DEST BASED ADDR) BYTE;
C0189     2          CC WHILE (CCLNT:=CCUNT -1)<>255;
C0190     2               DEST=CHAR;
0C191     3               ADDR=ADDR + 1;
CC192     3          ENC;
CC193     2     END FILL;
CC194     1
CC195     1          /*   *   *   *   *   *   SCANNER LITS *   *   *   *   */
CC196     1     CECLARE
0C197     1          LITERAL          LIT          '28';
0C198     1          INPUTSSTR        LIT          '47';
CC199     1          PERICD           LIT          '1';
00200     1          RPARIN           LIT          '3';
C0201     1          LPARIN           LIT          '2';
00202     1          INVALID          LIT          '0';
00203     1
00204     1
00205     1          /*   *   *   *   *   SCANNER TABLES *   *   *   *   */
00206     1     CECLARE TOKENSTABLE DATA
00207     1          /* CONTAINS THE TCKEN NUMBER ONE LESS THAN THE FIRST RESERVED WORC
C0208     1          FCR EACH LENGTH CF WCRD */
00209     1          (C,0,3,7,12,28,40,47,55,59,62),
C0210     1
```

130

```
00211    1        TABLE DATA('BY','GO','IF','TO','ADD','END','I-C'
00212    1             ,'NOT','RUN','CALL','ELSE','EXIT','FROM','INTO','LESS','MOVE'
00213    1             ,'NEXT','OPEN','PAGE','READ','SIZE','STOP','THRU','ZERO'
00214    1             ,'AFTER','CLOSE','ENTER','EQUAL','ERROR','INPUT','QUOTE','SPACE'
00215    1             ,'TIMES','UNTIL','USING','WRITE','ACCEPT','BEFORE','DELETE'
00216    1             ,'DIVIDE','OUTPUT','DISPLAY','GREATER'
00217    1             ,'INVALID','NUMERIC','PERFORM','REWRITE','ROUNDED','SECTION'
00218    1             ,'DIVISION','MULTIPLY','SENTENCE','SUBTRACT','ADVANCING',
00219    1             'DEPENDING','PROCEDURE','ALPHABETIC');
00220    1        OFFSET (11) ADDRESS INITIAL
00221    1             /*  NUMBER OF BYTES TO INDEX INTO THE TABLE FOR EACH LENGTH */
00222    1             (0,0,0,8,23,83,143,173,229,261,288),
00223    1
00224    1        WORD$COUNT DATA
00225    1             /* NUMBER OF WORDS OF EACH SIZE */
00226    1             (0,0,4,5,15,12,5,8,4,3,1),
00227    1
00228    1
00229    1        MAXSID$LEN          LIT          '12',
00230    1        MAX$LEN         .   LIT          '10',
00231    1        ADD$END             DATA ('END. '),
00232    1        LOCKED              BYTE INITIAL (0),
00233    1        HOLD                BYTE,
00234    1        BUFFER$END          ADDRESS   INITIAL   (100H),
00235    1        NEXT                BASED POINTER        BYTE,
00236    1        INBUFF              LIT          '80H',
00237    1        CHAR                BYTE      INITIAL(' '),
00238    1        ACCUM               BYTE,
00239    1        P$ACCUM             (30)         BYTE,
00240    1        DISPLAY             BYTE      INITIAL (0),
00241    1        DISPLAY$REST        (73)         BYTE,      /*RETURNED FROM SCANNER */
00242    1        TOKEN               BYTE;
00243    1
00244    1
00245    1        /*   PROCEDURES USED BY THE SCANNER */
00246    1
00247    1   NEXT$CHAR: PROCEDURE BYTE;
00248    2        IF LOCKED THEN
00249    2        DO;
00250    2           .  LOCKED=FALSE;
00251    3             RETURN (CHAR:=HOLD);
00252    3        END;
00253    2        IF (POINTER:=POINTER + 1) >= BUFFER$END THEN
00254    2        DO;
00255    2             IF NOT MORE$INPUT THEN
00256    3             DO;
00257    3                  BUFFER$END=.MEMORY;
00258    4                  POINTER=.ADD$END;
00259    4             END;
00260    3             ELSE POINTER=INBUFF;
00261    3        END;
00262    2        RETURN (CHAR:=NEXT);
00263    2   END NEXT$CHAR;
00264    1
00265    1   GET$CHAR: PROCEDURE;
00266    2        /* THIS PROCEDURE IS CALLED WHEN A NEW CHAR IS NEEDED WITHOUT
00267    2        THE DIRECT RETURN OF THE CHARACTER*/
00268    2        CHAR=NEXT$CHAR;
00269    2   END GET$CHAR;
00270    1
00271    1   DISPLAY$LINE: PROCEDURE;
00272    2        IF NOT LIST$INPUT THEN RETURN;
00273    2        DISPLAY(DISPLAY + 1) = '$';
00274    2        CALL PRINT(.DISPLAY$REST);
00275    2        DISPLAY=0;
00276    2   END DISPLAY$LINE;
00277    1
00278    1   LOAD$DISPLAY: PROCEDURE;
00279    2        IF DISPLAY<72 THEN
00280    2             DISPLAY(DISPLAY:=DISPLAY+1)=CHAR;
00281    2        CALL GET$CHAR;
00282    2   END LOAD$DISPLAY;
00283    1
00284    1   PUT: PROCEDURE;
00285    2        IF ACCUM < 30 THEN
00286    2        ACCUM(ACCUM:=ACCUM+1)=CHAR;
00287    2        CALL LOAD$DISPLAY;
00288    2   END PUT;
00289    1
00290    1   EAT$LINE: PROCEDURE;
00291    2        DO WHILE CHAR<>CR;
00292    2             CALL LOAD$DISPLAY;
00293    3        END;
00294    2   END EAT$LINE;
00295    1
00296    1   GET$NO$BLANK: PROCEDURE;
00297    2        DECLARE (N,I) BYTE;
00298    2        DO FOREVER;
00299    2             IF CHAR = ' ' THEN CALL LOAD$DISPLAY;
00300    3             ELSE
00301    3             IF CHAR=CR THEN
00302    3             DO;
00303    3                  CALL DISPLAY$LINE;
00304    4                  IF SEQ$NUM THEN N=8; ELSE N=2;
00305    4                  DO I = 1 TO N;
00306    4                       CALL LOAD$DISPLAY;
00307    5                  END;
00308    4                  IF CHAR = '*' THEN CALL EAT$LINE;
00309    4             END;
00310    3             ELSE
00311    3             IF CHAR = ':' THEN
00312    3             DO;
00313    3                  IF NOT DEBUGGING THEN CALL EAT$LINE;
00314    4                  ELSE
00315    4                  CALL LOAD$DISPLAY;
00316    4             END;
00317    3             ELSE
00318    3             RETURN;
00319    3        END;  /* END OF DO FOREVER */
00320    2   END GET$NO$BLANK;
```

131

```
00322   1    SPACE: PROCEDURE BYTE;
C0323   2        RETURN (CHAR=' ') CR (CHAR=CR);
C0324   2    END SPACE;
00325   1
00326   1    LEFTSPARIN: PRCCEDURE BYTE;
C0327   2        RETURN CHAR = '(';
C0328   2    END LEFTSPARIN;
C0329   1
C0330   1    RIGHTSPARIN: PRCCEDURE BYTE;
00331   2        RETURN CHAR = ')';
C0332   2    END RIGHTSPARIN;
00333   1
C0334   1    DELIMITER: PROCEDURE BYTE;
00335   2        /* CHECKS FCR A PERIOD FOLLCWED BY A SPACE CR CR*/
00336   2        IF CHAR <> '.' THEN RETURN FALSE;
C0337   2        HCLD=NEXTSCHAR;
C0338   2        LCCKED=TRUE;
C0339   2        IF SPACE THEN
C0340   2        DC;
00341   2            CHAR = '.';
C0342   3            RETURN TRUE;
C0343   3        END;
C0344   2        CHAR='.';
C0345   2        RETURN FALSE;
C0346   2    END DELIMITER;
00347   1
00348   1    ENDSCFSTOKEN: PRCCEDURE BYTE;
00349   2        RETURN SPACE OR  DELIMITER OR LEFTSPARIN OR RIGHTSPARIN;
C0350   2    END ENDSOFSTCKEN;
00351   1
C0352   1    GETSLITERAL: PRCCEDURE BYTE;
C0353   2        DC FOREVER;
C0354   2            IF NEXTSCHAR= CUOTE THEN RETURN LITERAL;
C0355   3            CALL PLT;
C0356   3        END;
C0357   2    END GETSLITERAL;
C0358   1
C0359   1
C0360   1    LCCKSLP: PRCCEDURE BYTE;
C0361   2        DECLARE PCINT ADDRESS,
00362   2        (HERE BASED PCINT,I) BYTE;
00363   2
C0364   2        MATCH: PRCCEDURE BYTE;
C0365   3            DECLARE J BYTE;
00366   3            DO J=1 TO ACCUM;
00367   3                IF HERE(J - 1) <> ACCUM(J) THEN RETURN FALSE;
C0368   4            END;
C0369   3            RETURN TRUE;
C0370   3        END MATCH;
00371   2
00372   2        PCINT=CFFSET(ACCUM)+ .TABLE;
C0373   2        DC I=1 TC HCFCSCOUNT(ACCUM);
C0374   2            IF MATCH THEN RETURN I;
C0375   3            PCINT = PCINT + ACCUM;
00376   3        END;
CC377   2        RETURN FALSE;
C0378   2    END LCCKSUP;
C0379   1
CC380   1    RESERVEDSWORD: PRCCEDURE BYTE;
CC381   2        /* RETURNS THE TCKEN NUMBER OF A RESERVED WORD IF THE CCNTENTS CF
00382   2        THE ACCUMULATCR IS A RESERVED WCRD, OTHERWISE RETURNS ZERO */
00383   2        DECLARE VALUE BYTE;
C0384   2        DECLARE NUMB BYTE;
C0385   2        IF ACCUM <= MAXSLEN THEN
00386   2        DC;
00387   2            IF (NLMB:=TCKENSTABLE(ACCUM))<>0 THEN
C0388   3            DC;
CC389   3                IF (VALUE:=LOOKSUP) <> 0 THEN
C0390   4                    NUMB=NUMB + VALUE;
OC391   4                ELSE NUMB=0;
CC392   4            END;
C0393   3        END;
CC394   2        RETURN NLMB;
CC395   2    END RESERVEDSWCRC;
CC396   1
CC397   1    GETSTCKEN: PRCCEDURE BYTE;
C0398   2        ACCUM=0;
C0399   2        CALL GETSNCSBLANK;
C0400   2        IF CHAR=CUCTE THEN RETURN GETSLITERAL;
00401   2        IF DELIMITER THEN
C0402   2        CC;
C0403   2            CALL PLT;
C0404   3            RETURN PERIOD;
00405   3        END;
C0406   2        IF LEFTSPARIN THEN
C0407   2        DC;
C0408   2            CALL PLT;
C0409   3            RETURN LPARIN;
C0410   3        END;
C0411   2        IF RIGHTSPARIN THEN
00412   2        DC;
00413   2            CALL PLT;
00414   3            RETURN RPARIN;
C0415   3        END;
C0416   2        DC FOREVER;
00417   2            CALL PLT;
C0418   2            IF ENDSCFSTOKEN THEN RETURN INPUTSSTR;
C0419   3        END; /* CF DC FOREVER */
C0420   2    END GETSTCKEN;
00421   1
00422   1        /*    ENC CF SCANNER ROUTINES   */
00423   1
00424   1        /*    SCANNER EXEC   */
C0425   1
00426   1    SCANNER: PRCCEDURE;
00427   2        IF(TCKEN:=GETSTCKEN) = INPUTSSTR THEN
C0428   2            IF (CTR:=RESERVEDSWORD) <> 0 THEN TOKEN=CTR;
C0429   2    END SCANNER;
C0430   1
C0431   1
C0432   1    PRINTSACCUM: PRCCEDURE;
00433   2        ACCUM(ACCUM+1)='$';
00434   2        CALL PRINT(.RSACCUM);
C0435   2    END PRINTSACCUM;
```

132

```
00437   1       PRINT$NUMBER: FRCCEDURE(NUMB);
C0438   2           CECLARE(NUMB,I,CNT,K) BYTE, J DATA(100,10);
C0439   2           CC I=0 TO 1;
00440   2               CNT=0;
00441   3               DC WHILE NUMB >= (K:=J(I));
00442   3                   NUMB=NUMB - K;
00443   4                   CNT=CNT + 1;
00444   4               END;
C0445   3               CALL FRINTCHAR('0' + CNT);
00446   3           END;
00447   2           CALL PRINTCHAR('0' + NUMB);
00448   2       END PRINT$NUMBER;
C0449   1
C0450   1
C0451   1
00452   1           /* * * * END OF SCANNER PROCEDURES * * * */
C0453   1
00454   1
00455   1           /* * * * * * SYMBOL TABLE CECLARATIONS * * * */
C0456   1
00457   1       CECLARE
C0458   1
C0459   1       CUR$SYM                ADCRESS,         /*SYMBOL BEING ACCESSEC*/
C0460   1       SYMBCL                 BASED CUR$SYM    BYTE,
00461   1       SYMBCL$ADDR            BASED CUR$SYM    ACCRESS,
00462   1       NE>T$SYM$ENTRY         BASED NEXT$SYM   ADCRESS,
00463   1       HASH$MASK              LIT              '3FH',
0C464   1       S$TYPE                 LIT              '2',
C0465   1       CISPLACEMENT           LIT              '12',
00466   1       CCCURS                 LIT              '11',
C0467   1       P$LENGTH               LIT              '3',
C0468   1       FLC$LENGTH             LIT              '3',
C0469   1       LEVEL                  LIT              '10',
C0470   1       REL$IC                 LIT              '5',
C0471   1       LCCATICN               LIT              '2',
C0472   1       START$NAME             LIT              '11',   /*1 LESS*/
00473   1       FCB$ADCR               LIT              '4',
C0474   1
00475   1
00476   1           /* * * * * * * SYMBOL TYPE LITERALS * * * * * * */
C0477   1
C0478   1
00479   1       UNRESCLVED             LIT              '255',
00480   1       LAEEL$TYPE             LIT              '32',
00481   1       MLLT$CCCURS            LIT              '128',
C0482   1       GRCUP                  LIT              '6',
CC483   1       NCN$NLMERICS$LIT       LIT              '7',
00484   1       ALFHA                  LIT              '8',
C0485   1       ALFHA$NUM              LIT              '9',
00486   1       LIT$SPACE              LIT              '10',
00487   1       LIT$CLCTE              LIT              '11',
C0488   1       LIT$ZERO               LIT              '12',
C0489   1       NUMERIC$LITERAL        LIT              '15',
C0490   1       NUMERIC                LIT              '16',
0C491   1       CCMP                   LIT              '21',
0C492   1       A$ED                   LIT              '72',
C0493   1       A$N$EC                 LIT              '73',
CC494   1       NUM$EC                 LIT              '80';
C0495   1
0C496   1
00497   1           /* * * * SYMBOL TABLE ROUTINES * * * */
CC498   1
CC499   1       SET$ADDRESS: PRCCEDURE(ACDR);
C0500   1       CECLARE ACCR ADDRESS;
C0501   2           SYMBOL$ADDR(LCCATION)=ADDR;
C0502   2       END SET$ADDRESS;
00503   1
00504   1       GET$ACCRESS: PRCCEDURE ADDRESS;
C0505   2           RETURN SYMBOL$ADCR(LCCATION);
00506   2       END GET$ADDRESS;
00507   1
00508   1       GET$FCB$ADCR: FRCCEDURE ADCRESS;
00509   2           RETURN SYMBCL$ADDR(FCB$ADDR);
C0510   2       ENC GET$FCB$ADCR;
C0511   1
00512   1       GET$TYPE: PRCCECLRE BYTE;
00513   2           RETURN SYMBCL(S$TYPE);
00514   2       END GET$TYPE;
C0515   1
00516   1       SET$TYPE: PRCCECLRE(TYPE);
00517   2           CECLARE TYPE BYTE;
00518   2           SYMBCL(S$TYPE)=TYPE;
C0519   2       END SET$TYPE;
C0520   1
00521   1       GET$LENGTH: FRCCEDURE ADDRESS;
00522   2           RETURN SYMBCL$ADCR(FLD$LENGTH);
00523   2       END GET$LENGTH;
00524   1
CC525   1       GET$LEVEL: FRCCECLRE BYTE;
00526   2           RETURN SFR(SYMBCL(LEVEL),4);
00527   2       ENC GET$LEVEL;
00528   1
CC529   1       GET$CECIMAL: PPCCEDURE BYTE;
C0530   2           RETURN SYMBCL(LEVEL) AND OFH;
00531   2       ENC GET$DEC$IMAL;
C0532   1
00533   1       GET$P$LENGTH: FRCCEDURE BYTE;
00534   2       RETURN SYMBCL(P$LENGTH);
00535   2       END GET$P$LENGTH;
C0536   1
00537   1       BLILD$SYMBCL: FRCCECURE(LEN);
00538   2           CECLARE LEN BYTE, TEMP ADDRESS;
C0539   2           TEMP=NE>T$SYM;
C0540   2           IF (NEXT$SYM:=.SYMBCL(LEN:=LEN + DISPLACEMENT))
00541   2               > MAX$MEMCRY THEN CALL FATAL$ERROR('ST');
C0542   2           CALL FILL (TEMP,0,LEN);
00543   2       ENC BLILD$SYMBCL;
00544   1
```

133

```
00545   1      ANDSCUTSCCCURS: PRDCECURE (TYPESIN) BYTE;
00546   2          CECLARE TYPESIN BYTE;
00547   2          RETURN TYPESIN  ANO 127;
00548   2      ENC ANDSCUTSCCCLRS;
C0549   1
C0550   1
00551   1          /*  *  *  *  PARSER DECLARATICNS  *  *  *  */
C0552   1      CECLARE
0C553   1      PSTACKSIZE      LIT         '30',      /* SIZE OF PARSE STACKS*/
0C554   1      VALUE           (PSTACKSIZE)  AODRESS,  /* TEMP VALUES */
CC555   1      STATESTACK      (PSTACKSIZE)  BYTE,   /* SAVEO STATES */
00556   1      VALUE2          (PSTACKSIZE)  ADDRESS,  /* VALUE2  STACK*/
00557   1      VARC            (100)         BYTE,    /*TEMP CHAR STCRE*/
00558   1      IDSSTACK        (20)          AODRESS,
C0559   1      IOSPTR          BYTE,
C0560   1      MAXSBYTE        BASED         MAXSINTSMEM         BYTE,
00561   1      SUBSIND         BYTE      INITIAL   (0),
00562   1      CCNDSTYPE       BYTE,
C0563   1      HCLOISECTICN    ACCRESS,
00564   1      HCLDISECSADCR   ACCRESS,
00565   1      SECTICNSFLAC    BYTE      INITIAL (0),
C0566   1      LSAOCR          ACCRESS,
C0567   1      LSLENGTH        ACCRESS,
C0568   1      LSTYPE          EYTE,
C0569   1      LSCEC           BYTE,
C0570   1      CCNSLENGTH      EYTE,
00571   1      CCMPILING       EYTE      INITIAL(TRUE),
00572   1      SP              BYTE      INITIAL (255),
00573   1      MP              EYTE,
00574   1      MPP1            EYTE,
00575   1      NCLCCK          EYTE      INITIAL(FALSE),
00576   1      (I,J,K)         BYTE,        /*INDICIES FDR THE PARSER*/
00577   1      STATE           EYTE      INITIAL(STARTS),
C0578   1
00579   1          /* * * * * * * * * CCOE LITERALS * * * * * * * * * */
C0580   1
C0581   1
00582   1          /* THE CCDE LITERALS ARE BRCKEN INTD GRDUPS CEPENDINC
00583   1           CN THE TCTAL LENGTH DF CCDE PRODUCED FOR THAT ACTICN */
C0584   1          /* LENGTH CNE */
C0585   1
00586   1      ACC LIT '1', /* REGISTER ADDITION */
00587   1      SUB LIT '2', /* REGISTER SUBTRACTICN */
C0588   1      MUL LIT '3', /* REGISTER MULTIPLICATION */
00589   1      DIV LIT '4', /* REGISTER DIVISION */
00590   1      NEG LIT '5', /* NOT CPERATOP */
00591   1      STP LIT '6', /* STOP PROGRAM */
00592   1      STI LIT '7', /* STCRE REGISTER 1 INTD REGISTER 0 */
0C593   1
0C5S4   1          /* LENGTH TWC */
00595   1      RND LIT '8', /* RDUNC CONTENTS OF REGISTER 1 */
0C596   1
0C597   1          /* LENCTH THREE */
00598   1      RET LIT '9', /* RETURN */
C0599   1      CLS LIT '1C',       /* CLOSE */
C0600   1      SEP LIT '11',       /* SIZE ERRDR */
00601   1      BRN LIT '12',       /* BRANCH */
00602   1      CFN LIT '13',       /* DPEN FOR INPUT */
00603   1      CP1 LIT '14',       /* OPEN FOR CUTPUT */
00604   1      OP2 LIT '15',       /* DPEN FDR I-O */
CC605   1      RGT LIT '16',       /* REGISTER GREATER THAN */
C0606   1      RLT LIT '17',       /* REGISTER LESS THAN */
C0607   1      REQ LIT '18',       /* REGISTER EQUAL */
C0608   1      INV LIT '19',       /* INVALID FILE ACTION */
CC609   1      EOR LIT '2C',       /* ENO CF FILE REACHEO */
C0610   1
00611   1          /* LENCTH FCLR */
00612   1      ACC LIT '21',       /* ACCEPT */
00613   1      DIS LIT '22',       /* OISPLAY */
00614   1      STO LIT '23',       /* STOP ANO DISPLAY */
00615   1      LDI LIT '24',       /* LDAO COUNTER IMEDIATE */
C0616   1
C0617   1          /* LENCTH FIVE */
00618   1      CEC LIT '25',       /* DECREMENT ANO BRANCH IF ZERO */
00619   1      STD LIT '26',       /* STDRE NUMERIC */
C0620   1      ST1 LIT '27',       /* STORE SIGNEO NUMERIC TRAILING */
00621   1      ST2 LIT '28',       /* STCRE SIGNED NUMERIC LEACING */
C0622   1      ST3 LIT '29',       /* STCRE SEPARATE SIGN LEACINC */
C0623   1      ST4 LIT '3C',       /* STCRE SEPARATE SIGN TRAILING */
00624   1      ST5 LIT '31',       /* STCRE CDMPUTATICNAL */
C0625   1
00626   1          /* LENGTH SIX */
C0627   1      LOD LIT '32',       /* LDAO NUMERIC LITERAL */
00628   1      LD1 LIT '33',       /* LDAO NUMERIC */
00629   1      LO2 LIT '34',       /* LDAO SIGNEO NUMERIC TRAILING */
C0630   1      LD3 LIT '35',       /* LCAO SIGNEO NUMERIC LEACING */
00631   1      LD4 LIT '36',       /* LCAD SEPARATE SIGN TRAILING */
00632   1      LO5 LIT '37',       /* LCAD SEPARATE SIGN LEADING */
00633   1      LDE LIT '38',       /* LCAD CDMPUTATICNAL */
C0634   1
00635   1          /* LENGTH SEVEN */
00636   1      PER LIT '39',       /* PERFORM */
00637   1      CNU LIT '4C',       /* CCMPARE FOR UNSIGNEO NUMERIC */
CC638   1      CNS LIT '41',       /* CCMPARE FOR SIGNED NUMERIC */
00639   1      CAL LIT '42',       /* CCMPARE FOR ALPHABETIC */
C0640   1      RWS LIT '43',       /* REWRITE SEQUENTIAL */
C0641   1      OLS LIT '44',       /* OELETE SECUENTIAL */
00642   1      ROF LIT '45',       /* REAO SEQUENTIAL */
00643   1      WTF LIT '46',       /* WRITE SECUENTIAL */
00644   1      RVL LIT '47',       /* REAO VARIABLE LENGTH */
00645   1      WVL LIT '48',       /* WRITE VARIABLE LENGTH */
C0646   1
00647   1          /* LENGTH NINE */
00648   1      SCR LIT '49',       /* SUBSCRIPT CCMPUTATION */
CC649   1      SGT LIT '5C',       /* STRING GREATER THAN */
CC650   1      SLT LIT '51',       /* STRING LESS THAN */
C0651   1      SEC LIT '52',       /* STRING EQUAL */
C0652   1      MGV LIT '53',       /* MCVE */
00653   1
```

```
00654    1          /* LENGTH 10 */
00655    1    RRS LIT  '54',      /* READ RELATIVE SEQUENTIAL */
00656    1    WRS LIT  '55',      /* WRITE RELATIVE SEQUENTIAL */
00657    1    RPR LIT  '56',      /* READ RELATIVE RANDOM */
00658    1    WRR LIT  '57',      /* WRITE RELATIVE RANDOM */
00659    1    RWR LIT  '58',      /* REWRITE RELATIVE */
00660    1    DLR LIT  '59',      /* DELETE RELATIVE */
00661    1
00662    1          /* LENGTH ELEVEN */
00663    1    MED LIT  '60',      /* MOVE EDITED */
00664    1
00665    1          /* LENGTH THIRTEEN */
00666    1    MNE LIT  '61',      /* MOVE NUMERIC EDITED */
00667    1
00668    1          /* VARIABLE LENGTH */
00669    1    GDP LIT  '62',      /* GO DEPENDING ON */
00670    1
00671    1          /* BUILD DIRECTING ONLY */
00672    1    INT LIT  '63',      /* INITIALIZE STORAGE */
00673    1    BST LIT  '64',      /* BACK STUFF ADDRESS */
00674    1    TER LIT  '65',      /* TERMINATE BUILD */
00675    1    SCD LIT  '66';      /* SET CODE START */
00676    1
00677    1          /*  *   *   *   PARSER ROUTINES   *   *   *   *   */
00678    1
00679    1    DIGIT: PROCEDURE (CHAR) BYTE;
00680    2        DECLARE CHAR BYTE;
00681    2        RETURN (CHAR<='9') AND (CHAR>='0');
00682    2    END DIGIT;
00683    1
00684    1    LETTER: PROCEDURE BYTE;
00685    2        RETURN (CHAR>='A') AND (CHAR<='Z');
00686    2    END LETTER;
00687    1
00688    1
00689    1    INVALID$TYPE: PROCEDURE;
00690    2        CALL PRINT$ERROR('IT');
00691    2    END INVALID$TYPE;
00692    1
00693    1    BYTE$OUT: PROCEDURE(ONE$BYTE);
00694    2        DECLARE ONE$BYTE BYTE;
00695    2        IF (OUTPUT$PTR:=OUTPUT$PTR + 1) > OUTPUT$END THEN
00696    2        DO;
00697    3            CALL WRITE$OUTPUT(.OUTPUT$BUFF);
00698    3            OUTPUT$PTR=.OUTPUT$BUFF;
00699    3        END;
00700    2        OUTPUT$CHAR=ONE$BYTE;
00701    2    END BYTE$OUT;
00702    1
00703    1    ADDR$OUT: PROCEDURE (ADDR);
00704    2        DECLARE ADDR ADDRESS;
00705    2        CALL BYTE$OUT(LOW(ADDR));
00706    2        CALL BYTE$OUT(HIGH (ADDR));
00707    2    END ADDR$OUT;
00708    1
00709    1    INC$COUNT: PROCEDURE(CNT);
00710    2        DECLARE CNT BYTE;
00711    2        IF(NEXT$AVAILABLE:=NEXT$AVAILABLE + CNT)
00712    2            >MAX$INT$MEM THEN CALL FATAL$ERROR('MC');
00713    2    END INC$COUNT;
00714    1
00715    1
00716    1
00717    1    ONE$ADDR$OPP: PROCEDURE(CODE,ADDR);
00718    2        DECLARE CODE BYTE, ADDR ADDRESS;
00719    2        CALL BYTE$OUT(CODE);
00720    2        CALL ADDR$OUT(ADDR);
00721    2        CALL INC$COUNT(3);
00722    2    END ONE$ADDR$OPP;
00723    1
00724    1    NOT$IMPLEMENTED: PROCEDURE;
00725    2        CALL PRINT$EROR ('NI');
00726    2    END NOT$IMPLEMENTED;
00727    1
00728    1    MATCH: PROCEDURE ADDRESS;
00729    2        /* CHECKS AN IDENTIFIER TO SEE IF IT IS IN THE SYMBOL
00730    2        TABLE. IF IT IS PRESENT, CUR$SYM IS SET FOR ACCESS,
00731    2        OTHERWISE THE POINTERS ARE SET FOR ENTRY*/
00732    2        DECLARE (POINT,COLLISION BASED POINT) ADDRESS, (HOLD,I) BYTE;
00733    2        IF VARC>MAX$IDS$LEN THEN
00734    2            VARC=MAX$IDS$LEN;
00735    2        HOLD=0;
00736    2        DO I=1 TO VARC;
00737    3            HOLD=HOLD+VARC(I);
00738    3        END;
00739    2        POINT=HASH$TAB$ADDR + SHL((HOLD AND HASH$MASK),1);
00740    2        DO FOREVER;
00741    2            IF COLLISION=0 THEN
00742    3                DO;
00743    3                    CUR$SYM,COLLISION=NEXT$SYM;
00744    4                    CALL BUILD$SYMBOL(VARC);
00745    4                    SYMBOL(P$LENGTH)=VARC;
00746    4                    DO I=1 TO VARC;
00747    4                        SYMBOL(START$NAME+I)=VARC(I);
00748    5                    END;
00749    4                    CALL SET$TYPE(UNRESOLVED); /* UNRESOLVED LABEL */
00750    4                    RETURN CUR$SYM;
00751    4                END;
00752    3                ELSE
00753    3                DO;
00754    3                    CUR$SYM=COLLISION;
00755    4                    IF (HOLD:=GET$P$LENGTH)=VARC THEN
00756    4                    DO;
00757    4                        I=I;
00758    5                        DO WHILE SYMBOL(START$NAME + I)= VARC(I);
00759    5                        IF (I:=I+I)>HOLD THEN RETURN(CUR$SYM:=COLLISION);
00760    6                        END;
00761    5                    END;
00762    4                END;
00763    3                POINT=COLLISION;
00764    3            END;
00765    2    END MATCH;
```

135

```
00767    1         SETSVALUE: FROCECURE(NUMB);
C0768    2             DECLARE NUMB ADDRESS;
C0769    2             VALUE(MF)=NUMB;
C0770    2         END SETSVALUE;
00771    1
00772    1         SETSVALUE2: FRCCEDURE(ADDR);
C0773    2             CECLARE ACCR ADDRESS;
C0774    2             VALUE2(MF)=ACDR;
00775    2         ENC SETSVALLE2;
00776    1
00777    1
C0778    1         SLBSCNT: PRCCECLRE BYTE;
00779    2             IF (SUBSINC:=SUBSIND + 1)>8 THEN
C0780    2                 SUBSINC=1;
C0781    2             RETURN SLBSIND;
00782    2         END SL8SCNT;
00783    1
C0784    1
C0785    1         CCCESBYTE: PRCCECURE (CODE);
C0786    2             CECLARE CODE BYTE;
C0787    2             CALL BYTESCLT(CODE);
C0788    2             CALL INCSCCLNT(1);
C0789    2         ENC CCCESBYTE;
CC790    1
0C791    1
0C792    1         CODESADCRESS: PRCCEDUPE (CODE);
0C793    2             CECLARE CCCE ADDRESS;
CC794    2             CALL ADCFSCLT(COCE);
C0795    2             CALL INCSCCLNT(2);
00796    2         ENC CCCESADDRESS;
0C797    1
CC798    1
CC799    1         INFLTSNUMERIC: FRCCECURE BYTE;
0C300    2             CC CTR=1 TC VARC;
00801    2                 IF NOT CIGIT(VARC(CTR)) THEN RETURN FALSE;
C0902    3             ENC;
CC803    2             FETURN TRUE;
00E04    2         END INFUTSNUMERIC;
C0E05    1
C0806    1
00E07    1         CCNVERTSINTEGER: PRCCEDURE ADDRESS;
00EC8    2             ACTR=0;
C02C9    2             CC CTR=1 TC VARC;
C0810    2                 IF NOT CIGIT(VARC(CTR)) THEN CALL PRINTSERROR('NN');
00811    3                 ASCTR=SFL(ACTR,3)+SHL(ACTR,1) + VARC(CTR) - '0';
C0812    3             ENC;
C0813    2             RETURN ACTR;
00E14    2         ENC CCNVERTSINTEGER;
00E15    1
00E16    1
00817    1         BACKSTLFF: PROCECURE (ACD1,ADD2);
C0E18    2             CECLARE (ACC1,ACC2) ADDRESS;
C0819    2             CALL BYTESCLT(BST);
C0920    2             CALL ADCFSCLT(ADC1);
J0E21    2             CALL ADCFSCLT(ADC2);
00822    2         ENC BACKSSTLFF;
CC822    1
C0824    1
00E25    1         UNRESOLVEDSBRANCF: PRCCEDURE;
00826    2             CALL SETSVALLE(NEXTSAVAILABE + 1);
00E27    2             CALL CNESACCRSOPP(BRN,0);
C0E28    2             CALL SETSVALLE2(NEXTSAVAILABLE);
C0829    2         END LNRESCLVECSRANCH;
C0830    1
00831    1
C0832    1         BACKSCCNC: FRCCECLRE;
00333    2             CALL BACKSTLFF(VALUE(SP-1),NEXTSAVAILABLE);
00834    2         ENC EACKSCONC;
C0E35    1
C0E36    1
JCE37    1         SETSBRANCF: FRCCECURE;
C0E38    2             CALL SETSVALLE(NEXTSAVAILABLE);
C0E39    2             CALL COCESACCRESS(0);
C0840    2         END SETSBRANCF;
J0E41    1
00E42    1
C0E43    1         KEEPSVALUES: PRCCEDURE;
00844    2             CALL SETSVALLE(VALUE(SP));
C0845    2             CALL SETSVALLE2(VALUE2(SP));
00346    2         ENC KEEPSVALLES;
00E47    1
00E48    1
C0849    1         STANCARCSATTRIBLTES: PROCEDURE(TYPE);
00E50    2             CECLARE TYPE BYTE;
00E51    2             CALL CCCESACCRESS(GETSFCBSADDR);
C0852    2             CALL CCCESACCRESS(GETSADDRESS);
C0853    2             CALL CCCESACCRESS(GETSLENGTH);
C0854    2             IF TYPE=0 THEN RETURN;
00855    2             CURSSYM=SYMBCLSACCR(RELSID);
00856    2             CALL CCCESACCRESS(GETSADDRESS);
C0857    2             CALL CCCESEYTE(GETSLENGTH);
C0858    2         END STANCARCSATTRIBUTES;
C0E59    1
C0E60    1
C0861    1         READSWRITE: PRCCECURE(INDEX);
00862    2             CECLARE INCEX BYTE;
C0E63    2
C0E64    2             IF (CTR:=CETSTYPE)=1 THEN
00E65    2             DC;
C0866    2                 CALL CCCESBYTE(RDF+INDEX);
C0E67    3                 CALL STANDARCSATTRIBUTES(0);
C0868    3             END;
C0869    2             ELSE IF CTR=2 THEN
CO870    2             CC;
00871    2                 CALL CCCESBYTE(RRS+INDEX);
00E72    3                 CALL STANDARCSATTRIBUTES(1);
OCE73    3             ENC;
C0874    2             ELSE IF CTR=3 THEN
C0E75    2             CC;
00E76    2                 CALL CCCESBYTE(RRR+INDEX);
00E77    3                 CALL STANCARDSATTRIBUTES(1);
COE78    3             END;
```

136

```
C0E79    2            ELSE IF CTR=4 THEN
CC880    2            CC;
J0881    2                  CALL CCCE$BYTE(RVL+INDEX);
00882    3                  CALL STANDARC$ATTRIBUTES(O);
00E83    3            END;
00E84    2            ELSE CALL FRINT$ERROR('FT');
C0E85    2      ENC READ$WRITE;
CCE86    1
00E87    1
CCE88    1      ARITHMETIC$TYPE: PRCCEDURE BYTE;
00889    2            IF ((L$TYPE:=AND$CUT$OCCURS(L$TYPE))>=NUMERIC$LITERAL)
CC890    2                  CR (L$TYPE<=CCMP) THEN RETURN L$TYPE - NUMERIC$LITERAL;
CC891    2            CALL INVALID$TYPE;
CC892    2            RETURN C;
CC893    2      END ARITHMETIC$TYPE;
C0E94    1
CCE95    1
C0E96    1      DEL$RWT: RRCCECLRE(FLAG);
OCE97    2            CECLARE FLAC BYTE;
OCE98    2            IF (CTR:=CET$TYPE)=3 THEN
C0E99    2            CC;
CU900    2                  IF FLAG THEN CALL CODE$BYTE(RWR);
OJS01    3                  ELSE CALL CODE$BYTE(DLR);
00S02    3                  CALL STANDARC$ATTRIBUTES(1);
COS03    3                  RETLRN;
J0S04    3            ENC;
COS05    2            IF (CTR=2) ANC (NOT FLAG) THEN CALL CODE$BYTE(DLS);
COS06    2            ELSE IF (CTR<>4) AND FLAG THEN CALL CODE$BYTE(RWS);
COS07    2            ELSE CALL INVALID$TYPE;
COS08    2            CALL STANDARC$ATTRIBUTES(O);
COS09    2      END DEL$RWT;
CCS1C    1
00S11    1
COS12    1      ATTRIELTES: PRCCEDURE;
J0S13    2            CALL CCDE$ADDRESS(L$ADCR);
COS14    2            CALL CODE$BYTE(L$LENGTH);
COS15    2            CALL CCDE$BYTE(L$DEC);
CCS16    2      END ATTRIBLTES;
00S17    1
COS18    1
COS19    1      LOAD$L$ID: PRCCECLRE(S$PTR);
COS20    2
00S21    2            CECLARE S$FTR BYTE;
COS22    2            IF((A$CTR:=VALUE(S$PTR))<NON$NUMERIC$LIT) CR
COS23    2                  (ACTR=NLMERIC$LITERAL) THEN
J0S24    2            CC;
COS25    2                  L$ADCR=VALUE2(SPTR);
COS26    3                  L$LENCTH=CCN$LENGTH;
OCS27    3                  L$TYPE=A$CTR;
COS28    3                  RETLRN;
COS29    3            ENC;
COS30    2            IF A$CTR<=LIT$ZERC THEN
00S31    2            CC;
00S32    2                  L$TYPE,L$ADCR=A$CTR;
COS33    3                  L$LENGTH=1;
OCS34    3                  RETLRN;
COS35    3            ENC;
COS36    2            CLR$SYM=VALLE(S$PTR);
00S37    2            L$TYPE=GET$TYPE;
COS38    2            L$LENGTH=GET$LENGTH;
COS39    2            L$CEC=GET$DECIMAL;
COS40    2            IF(L$ADCR:=VALUE2(S$PTR))=0 THEN L$ADCR=GET$ADDRESS;
COS41    2      END LCAD$L$ID;
COS42    1
COS43    1
COS44    1      LCAD$REG: PRCCECLRE(REG$NO,PTR);
COS45    2            CECLARE (REG$NO,PTR) BYTE;
CCS46    2            CALL LCAC$L$IC(PTR);
00S47    2            CALL CCDE$BYTE(LCC+ARITHMETIC$TYPE);
COS48    2            CALL ATTRIELTES;
COS49    2            CALL CCDE$BYTE(REG$NC);
COS50    2      END LCAD$REG;
CCS51    1
OCS52    1
COS53    1      STCRE$REG: PRCCEDURE(PTR);
00S54    2            CECLARE PTR BYTE;
00S55    2            CALL LCAC$L$IC(PTR);
00S56    2            CALL CCDE$BYTE(STC + ARITHMETIC$TYPE -1);
J0S57    2            CALL ATTRIELTES;
COS58    2      END STCRE$REG;
COS59    1
CCS60    1
CCS61    1      STCRE$CONSTANT: FRCCEDURE ADDRESS;
CCS62    2            IF(MAX$INT$MEM:=MAX$INT$MEM - VARC)<NEXT$AVAILABLE
00S63    2                  THEN CALL FATAL$ERROR('MC');
00S64    2            CALL BYTE$CLT(INT);
00S65    2            CALL ADCR$CLT(MAX$INT$MEM);
CCS66    2            CALL ADCR$CLT(CCN$LENGTH:=VARC);
00S67    2            DO CTR = 1 TC CCN$LENGTH;
COS68    2                  CALL BYTE$CLT(VARC(CTR));
COS69    3            ENC;
CCS70    2            RETURN MAX$INT$MEM;
COS71    2      ENC STCRE$CCNSTANT;
00S72    1
00S73    1
00S74    1      NUMERIC$LIT: PRCCEDURE BYTE;
CCS75    2            CECLARE CHAR BYTE;
CCS76    2            CC CTR=1 TC VARC;
00S77    2                  IF NCT( DIGIT(CHAR:=VARC(CTR))
COS78    3                        CR (CHAR='-') OR (CHAR='+')
COS79    3                        CR (CHAR='.')) THEN RETURN FALSE;
CCS80    3            ENC;
OCS81    2            RETURN TRLE;
CCS82    2      END NUMERIC$LIT;
CCS83    1
CCS84    1
OCS85    1      RCUND$STCRE: PRCCEDURE;
CCS86    2            IF VALUE(SP)<>O TFEN
OCS87    2            CC;
COS88    2                  CALL CCDE$BYTE(RND);
CCS89    2                  CALL CCDE$BYTE(L$DEC);
CC990    3            ENC;
COS91    2            CALL STCRE$REG(SP-1);
00S92    2      END RCUND$STCRE;
```

137

```
 00994    1
 00995    1     ADD$SUB: PRCCECLRE (INDEX);
 00996    2         CECLARE INDEX BYTE;
 00997    2         CALL LCAC$REC(0,MPP1);
 00998    2         IF VALUE(SP-3)<>0 THEN
 00999    2         CC;
 01000    2              CALL LCADSREG(1,SP-3);
 01001    3              CALL CCCESBYTE(ADD);
 01002    3              CALL CCCESBYTE(STI);
 01003    3         ENC;
 01004    2         CALL LCAC$REC(1,SP-1);
 01005    2         CALL CCCESEYTE(ADC + INDEX);
 01006    2         CALL RCUND$STCRE;
 01007    2     ENC ADC$SUB;
 01008    1
 01009    1
 01010    1     MULTSCIV: PRCCECLRE(INDEX);
 01011    2         CECLARE INCEX BYTE;
 01012    2         CALL LCAC$REC(0,MPP1);
 01013    2         CALL LCAC$REC(1,SP-1);
 01014    2         CALL CCCESBYTE(MUL + INDEX);
 01015    2         CALL RCUND$STCRE;
 01016    2     ENC MLLT$CIV;
 01017    1
 01018    1
 01019    1     CFECK$SUBSCRIPT: PROCEDURE;
 01020    2         CLR$SYM=VALLE(MP);
 01021    2         IF GET$TYPE<MLLT$CCCURS THEN
 01022    2         CC;
 01023    2              CALL PFINT$ERRCR('IS');
 01024    3              RETURN;
 01025    3         END;
 01026    2         IF INPLT$NUMERIC THEN
 01027    2         CC;
 01028    2              CALL SET$VALUE2(GET$ADDRESS + (GET$LENGTH * CCNVERT$INTEGER));
 01029    3              RETURN;
 01030    3         ENC;
 01031    2      ___CLR$SYM=MATCF;
 01032    2         IF ((CTR:=CET$TYPE)<NUMERIC) OR (CTR>COMP) THEN
 01033    2              CALL PRINT$ERRCR('TE');
 01034    2         CALL CNE$ADCR$OPP(SCR,GET$ADDRESS);
 01035    2         CALL CCCES$BYTE(SUB$CNT);
 01036    2         CALL CCCES$EYTE(GET$LENGTH);
 01037    2         CALL SET$VALLE2(SUB$IND);
 01038    2     ENC CFECK$SLESCRIPT;
 01039    1
 01040    1
 01041    1     LOAD$LABEL: FRCCECURE;
 01042    2         CLR$SYM=VALLE(MP);
 01043    2         IF (A$CTR:=GET$ADCRESS)<>0 THEN
 01044    2              CALL BACK$STUFF(A$CTR,VALUE2(MP));
 01045    2         CALL SET$ADCRESS(VALUE2(MP));
 01046    2         CALL SET$TYPE(LABEL$TYPE);
 01047    2         IF (A$CTR:=GET$FCB$ADCR)<>0 THEN
 01048    2              CALL BACK$STUFF(A$CTR,NEXT$AVAILABLE);
 01049    2         SYMBOL$ADCR(FCB$ADCR)=NEXT$AVAILABLE;
 01050    2         CALL CNE$ADCR$OPP(RET,0);
 01051    2     ENC LCAC$LAEEL;
 01052    1
 01053    1
 01054    1     LOAD$SEC$LABEL: FRCCECURE;
 01055    2         A$CTR=VALLE(MP);
 01056    2         CALL SET$VALLE(HOLD$SECTION);
 01057    2         FCLO$SECTICN=A$CTR;
 01058    2         A$CTR=VALUE2(MP);
 01059    2         CALL SET$VALLE2(HCLD$SEC$ADDR);
 01060    2         FCLO$SEC$ADCR = A$CTR;
 01061    2         CALL LCAC$LAEEL;
 01062    2     ENC LCAD$SEC$LAEEL;
 01063    1
 01064    1
 01065    1     LABEL$ADCR: FRCCECURE(ACDR,HOLD)ACDRESS;
 01066    2         CECLARE ACCR ACCRESS;
 01067    2         CECLARE FCLC BYTE;
 01068    2         CLR$SYM=ACCR;
 01069    2         IF(CTR:=CET$TYPE)=LABEL$TYPE THEN
 01070    2         CO;
 01071    2              IF FCLC THEN RETURN GET$ADDRESS;
 01072    3              RETURN CET$FCB$ADDR;
 01073    3         END;
 01074    2         IF CTR<>LNRESCLVEC THEN CALL INVALID$TYPE;
 01075    2         IF FOLC THEN
 01076    2         CC;
 01077    2              A$CTR=(CET$ADCRESS;
 01078    2              CALL SET$ADDRESS(NEXT$AVAILABLE + 1);
 01079    3              RETURN A$CTR;
 01080    3         ENC;
 01081    2         A$CTR=GET$FCB$ADCR;
 01082    2         SYMBOL$ADCR(FCB$ADDR)=NEXT$AVAILABLE + 1;
 01083    2         RETURN A$CTR;
 01084    2     END LABEL$ADCR;
 01085    1
 01086    1
 01087    1     CODE$FCP$DISFLAY: PROCEDURE (PCINT);
 01088    2         CECLARE FCINT BYTE;
 01089    2         CALL LCAC$L$IC(PCINT);
 01090    2         CALL CNE$ADCR$OPP(DIS,L$ADCR);
 01091    2         CALL CCCES$EYTE(L$LENGTH);
 01092    2     ENC CCDE$FOR$CISFLAY;
 01093    1
 01094    1
 01095    1     A$AN$TYPE: FRCCECURE EYTE;
 01096    2      RETURN (L$TYPE=ALPHA) OR (L$TYPE=ALPHA$NUM);
 01097    2     ENC A$AN$TYPE;
 01098    1
```

138

```
01C99   1          NOTSINTEGER: PROCECURE BYTE;
O1101   2              RETURN LSDEC<>0;
O1102   2          END NCTSINTEGER;
C1103   1
C1104   1
C1105   1          NUMERICSTYPE: FROCEDURE BYTE;
C1106   2              RETURN (LSTYFE>=NUMERIC) AND (LSTYPE<=COMP);
O1107   2          END NLMERICSTYPE;
O1108   1
C1109   1
C1110   1          GENSCCMPARE: PROCECURE;
O1111   2              CECLARE (HSTYPE,HSDEC) BYTE,
O1112   2                      (HSADDR,HSLENGTH) ADDRESS;
O1113   2
O1114   2              CALL LCADSLSIC(MP);
O1115   2              LSTYPE=ANDSCLTSOCCURS(LSTYPE);
O1116   2              IF CONDSTYPE=3 THEN   /* COMPARE FCR NUMERIC */
O1117   2              CD;
C1118   2                  IF ASANSTYPE DR (LSTYPE>COMP) THEN CALL INVALICSTYPE;
C1119   3                  IF LSTYPE=NUMERIC THEN CALL CODESBYTE(CNU);
C1120   3                  ELSE CALL COCESBYTE(CNS);
C1121   3                  CALL CCCESADCRESS(LSADDR);
O1122   3                  CALL CCCESADCRESS(LSLENGTH);
O1123   3                  CALL SETSERANCH;
O1124   3              END;
C1125   2              ELSE IF CCNDSTYPE=4 THEN
O1126   2              CC;
C1127   2                  IF NUMERICSTYPE THEN CALL INVALIDSTYPE;
C1128   3                  CALL CCCESBYTE(CAL);
C1129   3                  CALL CCCESADCRESS(LSADDR);
O1130   3                  CALL CCCESADCRESS(LSLENGTH);
O1131   3                  CALL SETSERANCH;
C1132   3              ENC;
O1133   2              ELSE DD;
O1134   3                  IF NUMERICSTYPE THEN CTR=1;
O1135   3                  ELSE CTR=0;
O1136   3                  HSTYPE=LSTYPE;
C1137   3                  HSDEC=LSDEC;
C1138   3                  HSADCR=LSADDR;
O1139   3                  HSLENGTH=LSLENGTH;
C1140   3                  CALL LCADSLSID(SP);
O1141   3                  IF NUMERICSTYPE THEN CTR=CTR+1;
O1142   3                  IF CTR=2 THEN   /* NUMERIC COMPARE */
O1143   3                  DO;
O1144   4                      CALL LCADSREG(O,MP);
C1145   4                      CALL LCADSREG(I,SP);
C1146   4                      CALL COCESBYTE(SUB);
C1147   4                      CALL CODESBYTE(RGT + CONDSTYPE);
C1148   4                      CALL SETSERANCH;
C1149   4                  ENC;
C1150   3                  ELSE CC;
C1151   3                      /* ALPHA NUMERIC CCMPARE */
C1152   3                      IF (HSDEC<>0) DR (HSTYPE=CDMP)
C1153   4                          DR (LSDEC<>0) CR (LSTYPE=CCMP)
C1154   4                          DR (HSLENGTH<>LSLENGTH) THEN CALL INVALICSTYPE;
C1155   4                      CALL CODESBYTE(SGT+CONDSTYPE);
C1156   4                      CALL CCCESADDRESS(HSADDR);
O1157   4                      CALL CCCESADDRESS(LSADDR);
O1158   4                      CALL CODESADDRESS(HSLENGTH);
C1159   4                  ENC;
C1160   3              ENC;
O1161   2          ENC GENSCOMPARE;
O1162   1
O1163   1
O1164   1          MCVESTYPE: PROCECURE BYTE;
C1165   2              CECLARE
C1166   2                  HOLDSTYPE BYTE,
C1167   2                  ALPHASNLMSMCVE        LIT '0',
C1168   2                  ASNSEDSMCVE           LIT '1',
C1169   2                  NLMERICSMCVE          LIT '2',
C1170   2                  NSEDSMCVE             LIT '3';
O1171   2
O1172   2              LSTYPE=ANDSCUTSOCCURS(LSTYPE);
O1173   2              IF((HOLDSTYPE:=ANDSCUTSOCCURS(GETSTYPE))=GRCUP) DR (LSTYPE=GROUP)
C1174   2                  THEN RETLRN ALPHASNUMSMCVE;
O1175   2              IF HOLDSTYPE=ALPHA THEN
O1176   2                  IF ASANSTYPE CR (LSTYPE=ASED) OR (LSTYPE=ASNSED)
C1177   2                      THEN RETURN ALPHASNUMSMOVE;
C1178   2              IF HOLDSTYPE=ALPHASNUM THEN
O1179   2              CC;
C1180   2                  IF NOTSINTEGER THEN CALL INVALIDSTYPE;
O1181   3                  RETLRN ALPHASNUMSMOVE;
C1182   3              ENC;
O1183   2              IF (HOLDSTYPE>=NUMERIC) AND (HOLDSTYPE<=CDMP) THEN
O1184   2              CC;
O1185   2                  IF (LSTYPE=ALPHA) DR (LSTYFE>COMP) THEN CALL INVALIDSTYPE;
O1186   3                  RETLRN NUMERICSMOVE;
C1187   3              ENC;
C1188   2              IF HOLDSTYPE=ASNSEC THEN
C1189   2              CC;
C1190   2                  IF NOTSINTEGER THEN CALL INVALIDSTYPE;
O1191   3                  RETLRN ASNSECSMCVE;
C1192   3              ENC;
O1193   2              IF HOLDSTYPE=ASED THEN
O1194   2                  IF ASANSTYPE OR (LSTYPE>CCMP) THEN RETURN ASNSEDSMCVE;
O1195   2              IF HOLDSTYPE=NUMSED THEN
O1196   2                  IF NUMERICSTYPE CR (LSTYPE=ALPHASNUM) THEN
O1197   2                      RETURN NSEDSMOVE;
C1198   2              CALL INVALICSTYPE;
C1199   2              RETURN C;
C1200   2          END MCVESTYPE;
C1201   1
```

139

```
01202   1
01203   1      GENSMOVE:PROCEDURE;
01204   2         DECLARE
01205   2            LENGTH1 ADDRESS,
01206   2            ADDR1 ADDRESS,
01207   2            EXTRA ADDRESS;
01208   2
01209   2         ADDSADDSLEN: PROCEDURE;
01210   3             CALL CODESADDRESS(ADDR1);
01211   3             CALL CODESADDRESS(LSADDR);
01212   3             CALL CODESADDRESS(LSLENGTH);
01213   3         END ADDSADDSLEN;
01214   2
01215   2         CODESFORSEDIT: PROCEDURE;
01216   3             CALL ADDSADDSLEN;
01217   3             CALL CODESADDRESS(GETSFC8SADDR);
01218   3             CALL CODESADDRESS(LENGTH1);
01219   3         END CODESFORSEDIT;
01220   2
01221   2         CALL LOADSLSIC(MPP1);
01222   2         CLRSSYM=VALUE(SP);
01223   2         IF (ADDR1:=VALUE2(SP))=0 THEN ADDR1=GETSADDRESS;
01224   2         LENGTH1=GETSLENGTH;
01225   2
01226   2         DO CASE MOVESTYPE;
01227   2
01228   2                 /* ALPHA NUMERIC MOVE */
01229   2
01230   2                 DO;
01231   3                     IF LENGTH1>LSLENGTH THEN EXTRA=LENGTH1-LSLENGTH;
01232   4                     ELSE DO;
01233   4                         EXTRA=0;
01234   5                         LSLENGTH=LENGTH1;
01235   5                     END;
01236   4                     CALL CODESBYTE(MOV);
01237   4                     CALL ADDSADDSLEN;
01238   4                     CALL CODESADDRESS(EXTRA);
01239   4                 END;
01240   3
01241   3                 /* ALPHA NUMERIC EDITED */
01242   3
01243   3                 DO;
01244   3                     CALL CODESBYTE(MED);
01245   4                     CALL CODESFORSEDIT;
01246   4                 END;
01247   3
01248   3                 /* NUMERIC MOVE */
01249   3
01250   3                 DO;
01251   3                     CALL LOADSREG(2,MPP1);
01252   4                     CALL STORESREG(SP);
01253   4                 END;
01254   3
01255   3                 /* NUMERIC EDITED MOVE */
01256   3
01257   3                 DO;
01258   3                     CALL CODESBYTE(MNE);
01259   4                     CALL CODESFORSEDIT;
01260   4                     CALL CODESBYTE(LSDEC);
01261   4                     CALL CODESBYTE(GETSDECIMAL);
01262   4                 END;
01263   3         END;
01264   2      END GENSMOVE;
01265   1
01266   1
01267   1
01268   1      CODESGEN: PROCEDURE(PRODUCTION);
01269   2         DECLARE PRODUCTION BYTE;
01270   2         IF PRINTSPROC THEN
01271   2         DO;
01272   3             CALL CRLF;
01273   3             CALL PRINTCHAR(PCUND);
01274   3             CALL PRINTSNUMBER(PRODUCTION);
01275   3         END;
01276   2
01277   2         DO CASE PRODUCTION;
01278   2
01279   2      /*  P R O D U C T I O N S */
01280   2
01281   2         /* CASE 0 NOT USED  */
01282   2                 :
01283   2
01284   3      /*       1   <P-DIV> ::= PROCEDURE DIVISION <USING> . <PROC-BODY>      */
01285   3
01286   3         DO;
01287   3             COMPILING = FALSE;
01288   4             IF SECTIONSFLAG THEN CALL LOADSSECSLABEL;
01289   4         END;
01290   3
01291   3      /*       2   <USING> ::= USING <ID-STRING>                              */
01292   3
01293   3         CALL NOTSIMPLIMENTED;    /* INTER PROG COMM */
01294   3
01295   3      /*       3                 <EMPTY>                                     */
01296   3
01297   3         ;   /* NO ACTION REQUIRED */
01298   3
01299   3      /*       4   <ID-STRING> ::= <ID>                                      */
01300   3
```

140

```
01301   3           ILSSIACK(IDSPIR.-U)=VALUE(SP);
01302   3   /*       5                   <ID-STRING> <ID>                    */
01303   3
01304   3       CC;
01305   3           IF(IDSPTR:=ICPTR+1)=20 THEN
01306   4           DO;
01307   5               CALL PRINTSERROR('ID');
01308   5               ICSPTR=19;
01309   5           ENC;
01310   4           IDSSTACK(IDSPTR)=VALUE(SP);
01311   4       END;
01312   3
01313   3   /*       6    <FRCC-EODY> ::= <PARAGRAPH>                         */
01314   3
01315   3       ;   /* NO ACTICN RECUIRED */
01316   3
01317   3   /*       7                   <PRCC-BODY> <PARAGRAPH>              */
01318   3
01319   3       ;   /* NO ACTICN RECUIRED */
01320   3
01321   3   /*       8    <PARAGRAPH> ::= <IC> . <SENTENCE-LIST>              */
01322   3
01323   3       CC;
01324   3           IF SECTICNSFLAG=0 THEN SECTIONSFLAG=2;
01325   4           CALL LCADSLABEL;
01326   4       END;
01327   3
01328   3   /*       9                   <ID> SECTION .                      */
01329   3
01330   3       CC;
01331   3           IF SECTICNSFLAG<>1 THEN
01332   4           DO;
01333   4               IF SECTIONSFLAG=2 THEN CALL PRINTSERROR('PF');
01334   5               SECTIONSFLAG=1;
01335   5               FCLCSSECTICN=VALUE(MP);
01336   5               FCLCSSECSADCR=VALUE2(MP);
01337   5           END;
01338   4           ELSE CALL LCADSSECSLABEL;
01339   4       ENC;
01340   3
01341   3   /*      10    <SENTENCE-LIST> ::= <SENTENCE> .                    */
01342   3
01343   3       ;   /* NO ACTICN RECUIRED */
01344   3
01345   3   /*      11                   <SENTENCE-LIST> <SENTENCE> .         */
01346   3
01347   3       ;   /* NO ACTICN RECUIRED */
01348   3
01349   3   /*      12    <SENTENCE> ::= <IMPERATIVE>                         */
01350   3
01351   3       ;   /* NO ACTICN REQUIRED */
01352   3
01353   3   /*      13                   <CONDITICNAL>                        */
01354   3
01355   3       ;   /* NO ACTICN RECUIRED */
01356   3
01357   3   /*      14                   ENTER <ID> <OPT-IC>                  */
01358   3
01359   3       CALL NOTSIMPLIMENTED;    /* LANGUAGE CHANGE */
01360   3
01361   3   /*      15    <IMPERATIVE> ::= ACCEPT <SUBID>                     */
01362   3
01363   3       CC;
01364   3           CALL LCADSLSIC(SP);
01365   4           CALL CNESADDRSCPP(ACC,LSADCR);
01366   4           CALL CCCESBYTE(LSLENGTH);
01367   4       END;
01368   3
01369   3   /*      16                   <ARITHMETIC>                         */
01370   3
01371   3       ;   /* NO ACTICN RECUIRED */
01372   3
01373   3   /*      17                   CALL <LIT> <USING>                   */
01374   3
01375   3       CALL NOTSIMPLIMENTEC;    /* INTER PROG COMM */
01376   3
01377   3   /*      18                   CLOSE <ID>                           */
01378   3
01379   3       CALL CNESADCRSDPP(CLS,GETSFCBSADCR);
01380   3
01381   3   /*      19                   <FILE-ACT>                           */
01382   3
01383   3       ;   /* NO ACTICN RECUIRED */
01384   3
01385   3   /*      20                   DISPLAY <LIT/ID> <OPT-LIT/IC>        */
01386   3
01387   3       CC;
01388   3           CALL CCDESFORSDISPLAY(MPP1);
01389   4           IF VALUE(SP)<>0 THEN CALL CODESFORSDISPLAY(SP);
01390   4       END;
01391   3
01392   3   /*      21                   EXIT <PROGRAM-IC>                    */
01393   3
01394   3       ;   /* NO ACTICN RECUIRED */
01395   3
01396   3   /*      22                   GO <ID>                              */
01397   3
01398   3       CALL CNESACCRSOPP(BRN,LABELSADDR(VALUE(SP),1));
01399   3
```

141

```
01400   3    /*      23                    GC <ID-STRING> CEPENDING <IC>           */
01401   3
01402   3        CC;
01403   4            CALL CCDESBYTE(GDP);
01404   4            CALL CCDESBYTE(IDSPTR);
01405   4            CURSSYM=VALUE(SP);
01406   4            CALL CCDESBYTE(GETSLENGTH);
01407   4            CALL CCDESADDRESS(GETSADDRESS);
01408   4            DC CTR=C TO ICSPTR;
01409   4                CALL CGDESADDRESS(LABELSADDR(IDSSTACK(IDSPTR),1));
01410   5            ENC;
01411   4        END;
01412
01413   3    /*      24                    MOVE <LIT/IO> TC <SUBID>               */
01414   3
01415   3        CALL GENSMCVE;
01416
01417   3    /*      25                    DPEN <TYPE-ACTICN> <ID>                */
01418   3
01419   3        CALL CNESACCRSCPP(CPN + VALUE(MPP1), GETSFCBSADCR);
01420
01421   3    /*      26                    PERFORM <ID> <THRU> <FINISH>           */
01422   3
01423   3        CC;
01424   3            CECLARE (ACDR2,ADDR3) ADDRESS;
01425   4            IF VALUE(SP-1)=O THEN ADDR2=LABELSADDR(VALUE(MPPI),0);
01426   4            ELSE ADDR2=LABELSADDR(VALUE(SP-1),0);
01427   4            IF (ACDR3:=VALUE2(SP)=O THEN ADDR3=NEXTSAVAILABLE + 7;
01428   4            ELSE CALL BACKSTUFF(VALUE(SP),NEXTSAVAILABLE + 7);
01429   4            CALL CNESADDRSCPP(PER,LABELSADDR(VALUE(MPPI),1));
01430   4            CALL CCDESADCRESS(ACDR2);
01431   4            CALL CCDESADCRESS(ADDR3);
01432   3        ENC;
01433   3
01434   3    /*      27                          <READ-IC>                        */
01435   3
01436   3        CALL NCTSIMPLIMENTEC;    /* GRAMMAR ERROR */
01437   3
01438   3    /*      28                    STOP <TERMINATE>                       */
01439   3
01440   3        CC;
01441   4            IF VALLE(SP)=O THEN CALL CODESBYTE(STP);
01442   4            ELSE CALL CNESACCRSDPP(STD,VALUE(SP));
01443   4        ENC;
01444   2
01445   3    /*      29   <CCNDITIONAL> ::= <ARITHMETIC> <SIZE-ERROR>           */
01446   3    /*      29                       <IMPERATIVE>                      */
01447   3
01448   3        CALL BACKSCCNC;
01449   3
01450   3    /*      30                    <FILE-ACT> <INVALID> <IMPERATIVE>     */
01451   3
01452   3        CALL BACKSCCND;
01453   3
01454   3    /*      31                    IF <CONDITICN> <ACTION> ELSE          */
01455   3    /*      31                       <IMPERATIVE>                       */
01456   3
01457   3        CC;
01458   3            CALL BACKSTUFF(VALUE(MPP1),VALUE2(SP-2));
01459   4            CALL BACKSTUFF(VALUE(SP-2),NEXTSAVAILABLE);
01460   4        ENC;
01461   3
01462   2    /*      32                    <READ-ID> <SPECIAL> <IMPERATIVE>      */
01463   3
01464   3        CALL BACKSCCND;
01465   3
01466   3    /*      33   <ARITHMETIC> ::= ADD <L/ID> <OPT-L/ID> TO <SLBID>     */
01467   3    /*      33                       <RCUND>                           */
01468   3
01469   3        CALL ADCSSUB(O);
01470   3
01471   3    /*      34                    DIVIDE <L/IC> INTC <SUBIC> <RCUND>    */
01472   3
01473   3        CALL MLLTSCIV(1);
01474   3
01475   3    /*      35                    MULTIPLY <L/ID> BY <SUBID> <RCUNC>    */
01476   3
01477   3        CALL MULTSDIV(0);
01478   3
01479   3    /*      36                    SUBTRACT <L/ID> <CPT-L/IC> FRCM       */
01480   3    /*      36                       <SUBIO> <ROUND>                    */
01481   3
01482   3        CALL ADCSSLE(1);
01483   3
01484   3    /*      37   <FILE-ACT> ::= DELETE <ID>                            */
01485   3
01486   3        CALL DELSRKT(O);
01487   3
01488   3    /*      38                    REWRITE <ID>                          */
01489   3
01490   3        CALL DELSRKT(1);
01491   3
01492   3    /*      39                    WRITE <ID> <SPECIAL-ACT>              */
01493   3
01494   3        CALL REACSWRITE(1);
01495   3
01496   3    /*      40   <CCNDITICN> ::= <LIT/ID> <NOT> <CCNC-TYPE>            */
01497   3
01498   3        CALL GENSCCMFARE;
01499   3
01500   3    /*      4I   <CCNC-TYPE> ::= NUMERIC                              */
01501   3
01502   3        CCNDSTYFE=3;
01503   3
01504   3    /*      42                    ALPHABETIC                            */
01505   3
01506   3        CCNDSTYPE=4;
01507   3
01508   3    /*      43                    <CCMPARE> <LIT/IC>                    */
01509   3
01510   3        CALL KFEFSVALUES;
```

142

```
01511   3    /*    44   <NOT> ::= NOT                                          */
01512   3
01513   3       CALL CODE$BYTE(NEG);
01514   3
01515   3
01516   3    /*    45            <EMPTY>                                       */
01517   3
01518   3       ;   /* NO ACTION REQUIRED */
01519   3
01520   3    /*    46   <COMPARE> ::= GREATER                                  */
01521   3
01522   3       CCNDSTYPE=0;
01523   3
01524   3    /*    47            LESS                                          */
01525   3
01526   3       CCNDSTYPE=1;
01527   3
01528   3    /*    48            EQUAL                                         */
01529   3
01530   3       CCNDSTYPE=2;
01531   3
01532   3    /*    49   <ROUND> ::= ROUNDED                                    */
01533   3
01534   3       CALL SET$VALUE(1);
01535   3
01536   3    /*    50            <EMPTY>                                       */
01537   3
01538   3       ;   /* NO ACTION REQUIRED */
01539   3
01540   3    /*    51   <TERMINATE> ::= <LITERAL>                              */
01541   3
01542   3       ;   /* NO ACTION REQUIRED */
01543   3
01544   3    /*    52            RUN                                           */
01545   3
01546   3       ;   /* NO ACTION REQUIRED - VALUE(SP) ALREADY ZERO */
01547   3
01548   3    /*    53   <SPECIAL> ::= <INVALID>                                */
01549   3
01550   3       ;   /* NO ACTION REQUIRED */
01551   3
01552   3    /*    54            END                                          */
01553   3
01554   3       CC;
01555   3          CALL SET$VALUE(2);
01556   4          CALL CODE$BYTE(EOR);
01557   4          CALL SET$BRANCH;
01558   4       END;
01559   3
01560   3    /*    55   <CPT-ID> ::= <SUBID>                                   */
01561   3
01562   3       ;   /* VALUE AND VALUE2 ALREADY SET */
01563   3
01564   3    /*    56                                                         */
01565   3
01566   3       ;   /* VALUE ALREADY ZERO */
01567   3
01568   3    /*    57   <ACTION> ::= <IMPERATIVE>                        . ,    */
01569   3
01570   3       CALL UNRESOLVED$BRANCH;
01571   3
01572   3    /*    58            NEXT SENTENCE                                 */
01573   3
01574   3       CALL UNRESOLVED$BRANCH;
01575   3
01576   3    /*    59   <THRU> ::= THRU <ID>                                   */
01577   3
01578   3       CALL KEEP$VALUES;
01579   3
01580   3    /*    60                                                         */
01581   3
01582   3       ;   /* NO ACTION REQUIRED */
01583   3
01584   3    /*    61   <FINISH> ::= <L/ID> TIMES                             */
01585   3
01586   3       CC;
01587   3.         CALL LOAD$L$ID(MP);
01588   4          CALL ONE$ADDR$OPP(LDI,L$ADDR);
01589   4          CALL CODE$BYTE(L$LENGTH);
01590   4          CALL SET$VALUE2(NEXT$AVAILABLE);
01591   4          CALL ONE$ADDR$OPP(DEC,0);
01592   4          CALL CODE$ADDRESS(0);
01593   4          CALL SET$VALUE(NEXT$AVAILABLE);
01594   4       END;
01595   3
01596   3    /*    62            UNTIL <CONDITION>                             */
01597   3
01598   3       CALL KEEP$VALUES;
01599   3
01600   3    /*    63                                                         */
01601   3
01602   3       ;  . /* NO ACTION REQUIRED */
01603   3
01604   3    /*    64   <INVALID> ::= INVALID                                  */
01605   3
01606   3       CC;
01607   3          CALL SET$VALUE(1);
01608   4          CALL CODE$BYTE(INV);
01609   4          CALL SET$BRANCH;
01610   4       END;
01611   3
01612   3    /*    65   <SIZE-ERROR> ::= SIZE ERROR                            */
01613   3
01614   3       CC;
01615   3          CALL CODE$BYTE(SER);
01616   3          CALL UNRESOLVED$BRANCH;
01617   4       END;
01618   3
```

```
01619   3      /*      66   <SPECIAL-ACT> ::= <WHEN> ADVANCING <HOW-MANY>        */
01620   3
01621   3           CALL NOTSIMPLIMENTED;    /* CARRAGE CCNTROL */
01622   3
01623   3      /*      67                                                        */
01624   3
01625   3           ;    /* NO ACTICN REQUIRED */
01626   3
01627   3      /*      68   <WHEN> ::= BEFORE                                     */
01628   3
01629   3           CALL NOTSIMPLIMENTED;    /* CARRAGE CCNTROL */
01630   3
01631   3      /*      69                 AFTER                                  */
01632   3
01633   3           CALL NCTSIMPLIMENTED;    /* CARRAGE CCNTROL */
01634   3
01635   3      /*      70   <HCW-MANY> ::= <INTEGER>                             */
01636   3
01637   3           CALL NOTSIMPLIMENTED;    /* CARRAGE CCNTROL */
01638   3
01639   3      /*      71                 PAGE                                   */
01640   3
01641   3           CALL NCTSIMPLIMENTED;    /* CARRAGE CCNTROL */
01642   3
01643   3      /*      72   <TYPE-ACTICN> ::= INPUT                              */
01644   3
01645   3           ;    /* NO ACTICN REQUIRED - VALUE(SP) ALREADY ZERC */
01646   3
01647   3      /*      73                 OUTPUT                                 */
01648   3
01649   3           CALL  SETSVALUE(1);
01650   3
01651   3      /*      74                 I-C                                    */
01652   3
01653   3           CALL SETSVALLE(2);
01654   3
01655   3      /*      75   <SUBID> ::= <SUBSCRIPT>                              */
01656   3
01657   3           ;    /* VALLE AND VALUE2 ALREADY SET */
01658   3
01659   3      /*      76                 <ID>                                   */
01660   3
01661   3           ;    /* NO ACTICN RECUIRED */
01662   3
01663   3      /*      77   <INTEGER> ::= <INPUT>                                */
01664   3
01665   3           CALL SETSVALUE(CCNVERTSINTEGER);
01666   3
01667   3      /*      78   <IC> ::= <INPUT>                                     */
01668   3
01669   3           CC;
01670   3              CALL SETSVALUE(MATCH);
01671   4              IF CETSTYPE=UNRESOLVED THEN CALL SETSVALUE2(NEXTSAVAILABLE);
01672   4           END;
01673   3
01674   3      /*      79   <L/ID> ::= <INPUT>                                   */
01675   3
01676   3           CC;
01677   3              IF NUMERICSLIT THEN
01678   4              DC;
01679   4                 CALL SETSVALUE(NUMERICSLITERAL);
01680   5                 CALL SETSVALUE2(STCRESCCNSTANT);
01681   5              ENC;
01682   4              ELSE CALL SETSVALUE(MATCH);
01683   4           END;
01684   3
01685   3      /*      80                 <SUBSCRIPT>                            */
01686   3
01687   3           ;    /* NO ACTICN RECUIRED */
01688   3
01689   3      /*      81                 ZERO                                   */
01690   3
01691   3           CALL SETSVALUE(LITSZERO);
01692   3
01693   3      /*      82   <SLBSCRIPT> ::= <IC> ( <INPUT> )                     */
01694   3
01695   3           CALL CHECKSSUBSCRIPT;
01696   3
01697   3      /*      83   <GFT-L/ID> ::= <L/ID>                                */
01698   3
01699   3           ;    /* NO ACTICN RECUIRED */
01700   3
01701   3      /*      84                 <EMPTY>                                */
01702   3
01703   3           ;    /* VALLE ALREADY SET */
01704   3
01705   3      /*      85   <NN-LIT> ::= <LIT>                              .    */
01706   3
01707   3           CC;
01708   3              CALL SETSVALUE(NONSNUMERICSLIT);
01709   4              CALL SETSVALUE2(STORESCCNSTANT);
01710   4           ENC;
01711   3
01712   3      /*      86                 SPACE                            .     */
01713   3
01714   3           CALL SETSVALUE(LITSSPACE);
01715   3
01716   3      /*      87                 QUOTE                                  */
01717   3
01718   3           CALL SETSVALUE(LITSQUOTE);
01719   3
01720   3      /*      88   <LITERAL> ::= <NN-LIT>                               */
01721   3
01722   3           ;    /* NO ACTION RECUIRED */
01723   3
```

144

```
01724   3    /*      89              <INPUT>                           */
01725   3
01726   3        CC:
01727   3            IF NOT NUMERIC$LIT THEN CALL INVALIC$TYPE;
01728   4            CALL SETSVALUE(NUMERIC$LITERAL);
01729   4            CALL SETSVALUE2(STORE$CCNSTANT);
01730   4        END;
01731   3
01732   3    /*      90              ZERO                              */
01733   3
01734   3        CALL SETSVALUE(LITSZERO);
01735   3
01736   3    /*      91    <LIT/ID> ::= <L/ID>                         */
01737   3
01738   3        ;   /* NO ACTION REQUIRED */
01739   3
01740   3    /*      92              <NN-LIT>                          */
01741   3
01742   3        ;   /* NO ACTICN REQUIRED */
01743   3
01744   3    /*      93    <CFT-LIT/ID> ::= <LIT/ID>                   */
01745   3
01746   3        ;   /* NO ACTICN REQUIRED */
01747   3
01748   3    /*      94                  <EMPTY>                       */
01749   3
01750   3        ;   /* NO ACTICN REQUIRED */
01751   3
01752   3    /*      95    <FFCGRAM-IC> ::= <ID>                       */
01753   3
01754   3        CALL NCTSIMPLIMENTED;    /* INTER PROG COMM */
01755   3
01756   3    /*      96                                                */
01757   3        ;   /* NO ACTICN REQUIRED */
01758   3
01759   3    /*      97    <PEAC-IO> ::= READ <IC>                     */
01760   3
01761   3        CALL REACS$WRITE(0);
01762   3
01763   3        ENC;    /* END OF CASE STATEMENT */
01764   2    END CCCESGEN;
01765   1
01766   1    GETIN1: PROCEDURE BYTE;
01767   2        RETURN INDEX1(STATE);
01768   2    ENC GETIN1;
01769   1
01770   1    GETIN2: FRCCEDURE BYTE;
01771   2        RETURN INCEX2(STATE);
01772   2    ENC GETIN2;
01773   1
01774   1    INCSP: PROCECURE;
01775   2        VALUE(SF:=SP + 1)=0;    /* CLEAR THE STACK WHILE INCREMENTING */
01776   2        VALUE2(SF)=C;
01777   2        IF SP >= PSTACKSIZE THEN CALL FATAL$ERROR('SC');
01778   2    ENC INCSP;
01779   1
01780   1    LCCKAHEAC: FRCCEDURE;
01781   2        IF NCLCCK THEN
01782   2        CC:
01783   3            CALL SCANNER;
01784   3            NCLCCK=FALSE;
01785   3            IF FRINTSTOKEN THEN
01786   3            DC:
01787   3                CALL CRLF;
01788   4                CALL PRINTSNUMBER(TOKEN);
01789   4                CALL PRINTSCHAR(' ');
01790   4                CALL PRINTSACCUM;
01791   4            ENC;
01792   3        ENC;
01793   2    END LCCKAHEAC;
01794   1
01795   1    NCSCCNFLICT: PRCCEDURE (CSTATE) BYTE;
01796   2        CECLARE (CSTATE,I,J,K) BYTE;
01797   2        J=INDEX1(CSTATE);
01798   2        K=J + INCEX2(CSTATE) - 1;
01799   2        CC I=J TC K;
01800   2            IF REAC1(I)=TCKEN THEN RETURN TRUE;
01801   3        ENC;
01802   2    RETURN FALSE;
01803   2    ENC NCSCCNFLICT;
01804   1
01805   1    RECOVER: PRCCECURE BYTE;
01806   2        CECLARE TSP BYTE, RSTATE BYTE;
01807   2        CC FCREVER;
01808   2            TSF=SP;
01809   3            DO WHILE TSP <> 255;
01810   3                IF NCSCCNFLICT(RSTATE:=STATESTACK(TSP)) THEN
01811   4                CC; /* STATE WILL READ TOKEN */
01812   4                    IF SP<>TSP THEN SP = TSP - 1;
01813   5                    RETURN RSTATE;
01814   5                ENC;
01815   4                TSF = TSP - 1;
01816   4            ENC;
01817   3            CALL SCANNER; /* TRY ANCTHER TOKEN */
01818   3        ENC;
01819   2    ENC RECCVER;
01820   1
```

```
01821    1          /* * * * * * PROGRAM EXECUTION STARTS HERE * * */
01822    1
01823    1
01824    1          /* INITIALIZATION */
01825    1
01826    1    TOKEN=62;        /* PRIME THE SCANNER WITH -PROCEDURE- */
01827    1    CALL MOVE(PASS1STOP-PASS1SLEN,.OUTPUTSFCB,PASS1SLEN);
01828    1        /* THIS SETS
01829    1            OUTPUT FILE CONTROL BLOCK
01830    1            TOGGLES
01831    1            READ POINTER
01832    1            NEXT SYMBOL TABLE POINTER
01833    1        */
01834    1    OUTPUTSEND=(OUTPUTSPTR:=.OUTPUTSBUFF-1)+128;
01835    1
01836    1        /* * * * * * * PARSER * * * * * */
01837    1
01838    1    DO WHILE COMPILING;
01839    1        IF STATE <=.MAXRNO THEN          /* READ STATE */
01840    2        DO;
01841    3            CALL INCSP;
01842    3            STATESTACK(SP) = STATE;   /* SAVE CURRENT STATE */
01843    3            CALL LOCKAHEAD;
01844    3            I=GETIN1;
01845    3            J = I + GETIN2 - 1;
01846    3            DO I=I TO J;
01847    3                IF READ1(I) = TOKEN THEN
01848    4                DO;
01849    4                /* COPY THE ACCUMULATOR IF IT IS AN INPUT
01850    4                STRING.  IF IT IS A RESERVED WORD IT DOES
01851    4                NOT NEED TO BE COPIED. */
01852    4                    IF (TOKEN=INPUTSSTR) OR (TOKEN=LITERAL) THEN
01853    5                        DO K=0 TO ACCUM;
01854    6                            VARC(K)=ACCUM(K);
01855    6                        END;
01856    5                    STATE=READ2(I);
01857    5                    NOLOCK=TRUE;
01858    5                    I=J;
01859    5                END;
01860    4                ELSE
01861    4                IF I=J THEN
01862    4                DO;
01863    4                    CALL PRINTSERROR('NP');
01864    5                    CALL PRINT(.' ERROR NEAR $');
01865    5                    CALL PRINTSACCUM;
01866    5                    IF (STATE:=RECOVER)=0 THEN COMPILING=FALSE;
01867    5                END;
01868    4            END;
01869    3        END;    /* END OF READ STATE */
01870    2        ELSE
01871    2        IF STATE>MAXRNO THEN          /* APPLY PRODUCTION STATE */
01872    2        DO;
01873    2            MP=SP - GETIN2;
01874    3            MPPI=MP + 1;
01875    3            CALL CODESGEN(STATE - MAXRNO);
01876    3            SP=MP;
01877    3            I=GETIN1;
01878    3            J=STATESTACK(SP);
01879    3            DO WHILE (K:=APPLY1(I)) <> 0 AND J<>K;
01880    4                I=I + 1;
01881    4            END;
01882    3            IF (K:=APPLY2(I))=0 THEN COMPILING=FALSE;
01883    3            STATE=K;
01884    3        END;
01885    2        ELSE
01886    2        IF STATE<=MAXLNO THEN        /*LOOKAHEAD STATE*/
01887    2        DO;
01888    2            I=GETIN1;
01889    3            CALL LOCKAHEAD;
01890    3            DO WHILE (K:=LOOK1(I))<>0 AND TOKEN <>K;
01891    3                I=I+1;
01892    4            END;
01893    3            STATE=LOOK2(I);
01894    3        END;
01895    2        ELSE
01896    2        DO;              /*PUSH STATES*/
01897    2            CALL INCSP;
01898    3            STATESTACK(SP)=GETIN2;
01899    3            STATE=GETIN1;
01900    3        END;
01901    2    END;    /* OF WHILE COMPILING */
01902    1    CALL BYTESOUT(TER);
01903    1    DO WHILE OUTPUTSPTR<>.OUTPUTSBUFF;
01904    1        CALL BYTESOUT(TER);
01905    2    END;
01906    1    CALL CLOSE;
01907    1    CALL CRLF;
01908    1    CALL PRINT(.'END OF PART 2   $');
01909    1    GO TO BOOT;
01910    1    EOF
```

146

```
00002    1            /*          CCBOL INTERPRETER        */
00C03    1
00C04    1    100H:     /* LOAC FOINT */
C0C05    1
C0006    1        /* GLCBAL CECLARATICNS AND LITERALS    */
00C07    1
G0C08    1    CECLARE
C0C0S    1
C0C10    1    LIT         LITERALLY        'LITERALLY',
00011    1    BDGS            LIT          '5H',        /* ENTRY TO CPERATING SYSTEM */
C0C12    1    BCCT            LIT          '0',
C0C13    1    CR              LIT          '13',
C0014    1    LF              LIT          '10',
C0C15    1    TRUE            LIT          '1',
C0C16    1    FALSE           LIT          '0',
C0C17    1    FOREVER         LIT          'WHILE TRUE';
C0C18    1
C0C19    1        /* UTILITY VARIABLES */
C0C20    1
C0C21    1    CECLARE
C0C22    1
00C23    1    INCEX           BYTE,
00C24    1    ASCTR           ACCRESS,
C0C25    1    CTR             BYTE,
CCC26    1    BASE            ACCRESS,
CCC27    1    BSBYTE          EASED BASE          BYTE,
C0C28    1    BSACCR          EASED BASE          ADDRESS,
C0C29    1    HCLD            ACCRESS,
C0030    1    HSBYTE          BASED HOLD          BYTE,
00031    1    HSACCR          EASED HCLD          ADDRESS,
00C32    1
C0C33    1
00C34    1        /* CCDE FCINTERS */
00C35    1
C0C36    1    CODESSTART              LIT          '2000H',
00C37    1    PRCCRAMSCCUNTEP         ADCRESS,
C0038    1    CSBYTE                  BASED PRCGRAMSCOUNTER     BYTE,
C0C39    1    CSADCR                  BASED PRDGRAMSCOUNTER     ACCRESS;
C0C40    1
C0C41    1
C0C42    1        /* * * * *    GLCBAL INPUT AND CUTPUT ROUTINES * * * * */
C0C43    1
C0C44    1
C0C45    1    CECLARE
C0C46    1    CURRENTSFCB ACCRESS,
C0C47    1    STARTSCFFSET      LIT          '36';
C0C48    1
C0C49    1    MCN1: PRCCECLRE (F,A);
C0C50    2        CECLARE F BYTE, A ACCRESS;
00C51    2        GC TC BCCS;
C0C52    2    END MCN1;
C0C53    1
00C54    1    MCN2: FROCECLRE (F,A)BYTE;
00C55    2        CECLARE F BYTE, A ACCRESS;
00C56    2        GC TC BCCS;
00C57    2    END MCN2;
C0C58    1
CCC59    1    PRINTSCHAR: PRCCECLRE (CHAR);
OCC60    2        CECLARE CHAR BYTE;
C0C61    2        CALL MCN1 (2,CHAR);
C0C62    2    END PRINTSCHAR;
00C63    1
00C64    1    CRLF: PRCCECLRE;
00C65    2        CALL PRINTSCHAR(CR);
00C66    2        CALL PRINTSCHAR(LF);
00C67    2    END CRLF;
C0C68    1
C0C69    1    PRINT: PROCECLRE (A);
C0070    2        CECLARE A ACCRESS;
0CC71    2        CALL CRLF;
C0C72    2        CALL MCN1(S,A);
0CC73    2    END PRINT;
00C74    1
CCC75    1
00C76    1    READ: PRCCECLRE(A);
C0C77    2        CECLARE A ACCRESS;
CCC78    2        CALL MCN1(IC,A);
C0C79    2    END READ;
CCC80    1
CCC81    1
0CC82    1    PRINTSERRCP: PRCCECLRE (CODE);
OCC83    2        CECLARE CCCE ADDRESS;
0CC84    2        CALL CRLF;
CCC85    2        CALL PRINTSCHAR(HIGH(CODE));
CCC86    2        CALL PRINTSCHAR(LCW(CODE));
OCC87    2    END PRINTSERRCR;
C0C88    1
C0C89    1
CCC90    1    FATALSERROR: PRCCEDURE(CCDE);
0CC91    2        CECLARE CCCE ADDRESS;
CCC92    2        CALL PRINTSERRCR(CDCE);
C0C93    2        CALL TIME(IC);
0CC94    2        /* CEBUG
OCC95    2        GC TC BCCT;
CCC96    2        DEBUG */
C0C97    2    END FATALSERPOR;
CCC98    1
CCC9S    1
C0100    1    OPEN: PRCCECLRE (AUCR) BYTE;
C0C101   2        CECLARE ACCR ADDRESS;
C0102    2        RETURN MCN2(15,ADCR);
C0103    2    ENC CPEN;
C0104    1
C0105    1
C0106    1    CLCSE: PROCECLRE (ACCR);
00107    2        CECLARE ACCR ACCRESS;
C0108    2        IF MCN2(16,ACCR)<>0 THEN CALL FATALSERROR('CL');
C0109    2    END CLCSE;
C0110    1
```

147

```
00111   1
00112   1     DELETE: PROCEDURE;
00113   2        CALL MON1(19,CURRENT$FCB);
00114   2     END DELETE;
00115   1
00116   1
00117   1     MAKE: PROCEDURE (ADDR);
00118   2        DECLARE ADDR ADDRESS;
00119   2        IF MON2(22,ADDR)<>0 THEN CALL FATAL$ERROR('ME');
00120   2     END MAKE;
00121   1
00122   1
00123   1     SET$DMA: PROCEDURE;
00124   2        CALL MON1(26,CURRENT$FCB+ START$OFFSET);
00125   2     END SET$DMA;
00126   1
00127   1
00128   1     DISK$READ: PROCEDURE BYTE;
00129   2        RETURN MON2(20,CURRENT$FCB);
00130   2     END DISK$READ;
00131   1
00132   1
00133   1     DISK$WRITE: PROCEDURE BYTE;
00134   2        RETURN MON2(21,CURRENT$FCB);
00135   2     END DISK$WRITE;
00136   1
00137   1
00138   1        /* * * * * * * * * * UTILITY PROCEDURES * * * * * * * * * * * * * */
00139   1
00140   1
00141   1     DECLARE
00142   1     SUBSCRIPT            (8)         ADDRESS;
00143   1
00144   1
00145   1     RES: PROCEDURE(ADDR) ADDRESS;
00146   2        /* THIS PROCEDURE RESOLVES THE ADDRESS OF A SUBSCRIPTED
00147   2        IDENTIFIER OR A LITERAL CONSTANT */
00148   2
00149   2        DECLARE ADDR ADDRESS;
00150   2        IF ADDR > 32 THEN RETURN ADDR;
00151   2        IF ADDR < 9 THEN RETURN SUBSCRIPT(ADDR);
00152   2        DO CASE ADDR - 9;
00153   3           RETURN .'0';
00154   3           RETURN .' ';
00155   3           RETURN .' ';
00156   3        END;
00157   2        RETURN 0;
00158   2     END RES;
00159   1
00160   1
00161   1     MOVE: PROCEDURE(FROM,DESTINATION,COUNT);
00162   2        DECLARE (FROM,DESTINATION,COUNT) ADDRESS,
00163   2           (F BASED FROM, D BASED DESTINATION) BYTE;
00164   2        DO WHILE (COUNT:=COUNT - 1) <> 0FFFFH;
00165   3           D=F;
00166   3           FROM=FROM + 1;
00167   3           DESTINATION=DESTINATION + 1;
00168   3        END;
00169   2     END MOVE;
00170   1
00171   1
00172   1     FILL: PROCEDURE(DESTINATION,COUNT,CHAR);
00173   2        DECLARE (DESTINATION,COUNT) ADDRESS,
00174   2
00175   2           (CHAR,D BASED DESTINATION) BYTE;
00176   2        DO WHILE (COUNT:=COUNT - 1)<> 0FFFFH;
00177   3           D=CHAR;
00178   3           DESTINATION=DESTINATION + 1;
00179   3        END;
00180   2     END FILL;
00181   1
00182   1
00183   1     CONVERT$TO$HEX: PROCEDURE(POINTER,COUNT) ADDRESS;
00184   2        DECLARE POINTER ADDRESS, COUNT BYTE;
00185   2        ASCTR=0;
00186   2        BASE=POINTER;
00187   2        DO CTR = 0 TO COUNT;
00188   2           ASCTR=SHL(ASCTR,3) + SHL(ASCTR,1) + B$BYTE(CTR) - '0';
00189   3        END;
00190   2        RETURN ASCTR;
00191   2     END CONVERT$TO$HEX;
00192   1
00193   1
00194   1        /* * * * * * * * * * * CODE CONTROL PROCEDURES * * * * * * * * * */
00195   1
00196   1     DECLARE
00197   1
00198   1     BRANCH$FLAG          BYTE         INITIAL(TRUE);
00199   1
00200   1     INC$PTR: PROCEDURE (COUNT);
00201   2        DECLARE COUNT BYTE;
00202   2        PROGRAM$COUNTER=PROGRAM$COUNTER + COUNT;
00203   2     END INC$PTR;
00204   1
00205   1
00206   1     GET$OP$CODE: PROCEDURE BYTE;
00207   2        CTR=C$BYTE;
00208   2        CALL INC$PTR(1);
00209   2        RETURN CTR;
00210   2     END GET$OP$CODE;
00211   1
```

148

```
00212   1              CCNDITIONAL$ERANCH: PROCEOURE(CCUNT);
00213   1                  /* THIS PRCCEOURE CONTROLS BRANCHING INSTRUCTIONS */
00214   2                  CECLARE CCUNT BYTE;
00215   2                  IF NCT BRANCH$FLAG THEN
00216   2                  CC;
00217   2
00218   3                      BRANCH$FLAG=TRUE;
00219   3                      PROGRAM$COUNTER=C$ADDR(CCUNT);
00220   3                  ENC;
00221   2                  ELSE CALL INC$PTR(SHL(CCUNT,1)+2);
00222   2              END CCNDITICNAL$ERANCH;
00223   1
00224   1
00225   1              INCREMENT$CR$BR$NCH: PROCEDURE(MARK);
00226   2                  CECLARE MARK BYTE;
00227   2                  IF MARK THEN CALL INC$PTR(2);
00228   2                  ELSE PROGRAM$COUNTER=C$ADDR;
00229   2              END INCREMENT$CF$ERANCH;
00230   1
00231   1                  /* * * * * * * * * * *CCMPARISONS * * * * * * * * * * * * * * * * */
00232   1
00233   1
00234   1
00235   1
00236   1              CHAR$CCMPARE: FRCCECURE BYTE;
00237   2                  BASE=C$ADCR;
00238   2                  HC LO=C$ADCR(1);
00239   2                  DO ASCTR=1 TC C$ADDR(2) - 1;
00240   2                      IF B$BYTE(ASCTR) > H$BYTE(ASCTR) THEN RETURN 0;
00241   3                      IF B$BYTE(ASCTR) < H$BYTE(ASCTR) THEN RETURN 1;
00242   3                  ENC;
00243   2                  RETURN 2;
00244   2              ENC CHAR$CCMPARE;
00245   1
00246   1
00247   1              STRING$CCMPARE: FRCCEDURE(PIVOT);
00248   2                  CECLARE PIVCT BYTE;
00249   2                  IF CHAR$CCMPARE<>PIVOT THEN BRANCH$FLAG=NCT ERANCH$FLAG;
00250   2                  CALL CCNDITICNAL$ERANCH(3);
00251   2              END STRING$CCMPARE;
00252   1
00253   1
00254   1              NUMERIC: PRCCECLRE(CHAR) BYTE;
00255   2                  CECLARE CHAR BYTE;
00256   2                  RETURN (CHAR >='0') AND (CHAR <='9');
00257   2              ENC NLMERIC;
00258   1
00259   1
00260   1              LETTER: PROCEOURE(CHAR) BYTE;
00261   2                  CECLARE CHAR BYTE;
00262   2                  RETURN (CHAR >='A') AND (CHAR <='Z');
00263   2              ENO LETTER;
00264   1
00265   1
00266   1              SIGN: PROCECLRE(CHAR) BYTE;
00267   2                  DECLARE CHAR BYTE;
00268   2                  RETURN (CHAR='+') OR (CHAR='-');
00269   2              ENO SIGN;
00270   1
00271   1
00272   1              CCMP$NLM$UNSIGNED: PRCCECURE;
00273   2                  BASE=C$ADCR;
00274   2                  CC ASCTR=0 TC C$ADDR(2)-1;
00275   2                      IF NOT NUMERIC(B$BYTE(ASCTR)) THEN
00276   3                      DO;
00277   3                          ERANCH$FLAG=NOT BRANCH$FLAG;
00278   4                          RETLRN;
00279   4                      ENC;
00280   3                  ENC;
00281   2                  CALL CCNCITICNAL$BRANCH(2);
00282   2              ENC CCMP$NUM$LNSIGNEO;
00283   1
00284   1
00285   1              CCMP$NUM$SICN: FRCCEDURE;
00286   2                  BASE=C$ADCR;
00287   2                  CC ASCTR=0 TC C$ADDR(2)-1;
00288   2                      IF NOT(NLMERIC(CTR:=B$BYTE(ASCTR))
00289   3                          CR SIGN(CTR)) THEN
00290   3                      DO;
00291   3                          ERANCH$FLAG=NOT BRANCH$FLAG;
00292   4                          RETURN;
00293   4                      ENC;
00294   3                  ENC;
00295   2                  CALL CCNCITICNAL$BRANCH(2);
00296   2              ENC CCMP$NUM$SICN;
00297   1
00298   1
00299   1              CCMP$ALPHA: FRCCEDURE;
00300   2                  BASE=C$ADCR;
00301   2                  CC ASCTR=0 TC C$ADDR(2)-1;
00302   2                      IF NOT LETTER(B$BYTE(ASCTR)) THEN
00303   3                      DO;
00304   3                          ERANCH$FLAG=NOT BRANCH$FLAG;
00305   4                          RETURN;
00306   4                      ENC;
00307   3                  ENO;
00308   2                  CALL CCNCITICNAL$BRANCH(2);
00309   2              ENC CCMP$ALPHA;
00310   1
00311   1
```

```
00312   1
00313   1              /* * * * * * * * * * *NUMERIC OPERATIONS * * * * * * * * * * */
00314   1
00315   1
00316   1      DECLARE
00317   1
00318   1      (R0,R1,R2)              (10)        BYTE, /* REGISTERS */
00319   1      (SIGN0,SIGN1,SIGN2)                BYTE,
00320   1      (DECSPT0,DECSPT1,DECSPT2)          BYTE,
00321   1      OVERFLOW                 BYTE,
00322   1      PSPTR                    BYTE,
00323   1      SWITCH                   BYTE,
00324   1      SIGNIF$NO                BYTE,
00325   1      ZERO$RESULT              BYTE,
00326   1      ZONE                     LIT        '10H',
00327   1      POSITIVE                 LIT        '1',
00328   1      NEGITIVE                 LIT        '0';
00329   1
00330   1
00331   1      CHECK$FOR$SIGN: PROCEDURE(CHAR) BYTE;
00332   2          DECLARE CHAR BYTE;
00333   2          IF NUMERIC(CHAR) THEN RETURN POSITIVE;
00334   2          IF NUMERIC(CHAR - ZONE) THEN RETURN NEGITIVE;
00335   2          CALL PRINT$ERROR('SI');
00336   2          RETURN POSITIVE;
00337   2      END CHECK$FOR$SIGN;
00338   1
00339   1
00340   1      STORE$IMMEDIATE: PROCEDURE;
00341   2          DO CTR=0 TO 9;
00342   2              R0(CTR)=R2(CTR);
00343   3          END;
00344   2          DECSPT0=DECSPT2;
00345   2          SIGN0=SIGN2;
00346   2      END STORE$IMMEDIATE;
00347   1
00348   1
00349   1      ONE$LEFT: PROCEDURE;
00350   2          DECLARE FLAG BYTE;
00351   2          IF ((FLAG:=SHR(B$BYTE,4))=0) OR (FLAG=9) THEN
00352   2          DO;
00353   2              DO CTR=0 TO 8;
00354   3                  B$BYTE(CTR)=SHL(B$BYTE(CTR),4) OR SHR(B$BYTE(CTR + 1),4);
00355   4              END;
00356   3              B$BYTE(9)=SHL(B$BYTE(9),4) OR FLAG;
00357   3          END;
00358   2          ELSE OVERFLOW=TRUE;
00359   2      END ONE$LEFT;
00360   1
00361   1
00362   1      ONE$RIGHT: PROCEDURE;
00363   2          CTR=10;
00364   2          DO INDEX=1 TO 9;
00365   2              CTR=CTR-1;
00366   3              B$BYTE(CTR)=SHR(B$BYTE(CTR),4) OR SHL(B$BYTE(CTR-1),4);
00367   3          END;
00368   2          B$BYTE=SHR(B$BYTE,4);
00369   2      END ONE$RIGHT;
00370   1
00371   1
00372   1      SHIFT$RIGHT: PROCEDURE(COUNT);
00373   2          DECLARE COUNT BYTE;
00374   2          DO CTR=1 TO COUNT;
00375   2              CALL ONE$RIGHT;
00376   3          END;
00377   2      END SHIFT$RIGHT;
00378   1
00379   1
00380   1      SHIFT$LEFT: PROCEDURE (COUNT);
00381   2          DECLARE COUNT BYTE;
00382   2          OVERFLOW=FALSE;
00383   2          DO CTR=1 TO COUNT;
00384   2              CALL ONE$LEFT;
00385   3              IF OVERFLOW THEN RETURN;
00386   3          END;
00387   2      END SHIFT$LEFT;
00388   1
00389   1
00390   1      ALLIGN: PROCEDURE;
00391   2          BASE=.R0;
00392   2          IF DECSPT0 > DECSPT1 THEN CALL SHIFT$RIGHT(DECSPT0-DECSPT1);
00393   2          ELSE CALL SHIFT$LEFT(DECSPT1-DECSPT0);
00394   2      END ALLIGN;
00395   1
```

150

```
00356    1       ADD$RO: PROCEDURE(SECOND, DEST);
00397    1           DECLARE (SECOND, DEST) ADDRESS, (CY,A,B,I) BYTE;
00398    2           HOLD= SECOND;
00399    2           BASE = DEST;
00400    2           CY=0;
00401    2           CTR=9;
00402    2           DO INDEX=1 TO 10;
00403    2               A=RO(CTR);
00404    2               B=H$BYTE(CTR);
00405    3               I=DEC(A+CY);
00406    3               CY=CARRY;
00407    3               I=DEC(I + B);
00408    3               CY=(CY OR CARRY) AND 1;
00409    3               B$BYTE(CTR)=I;
00410    3               CTR=CTR-1;
00411    3           END;
00412    3           IF CY THEN
00413    2           DO;
00414    2               CTR=9;
00415    2               DO INDEX = 1 TO 10;
00416    3                   I=R2(CTR);
00417    3                   I=DEC(I+CY);
00418    4                   CY=CARRY AND 1;
00419    4                   R2(CTR)=I;
00420    4                   CTR=CTR-1;
00421    4               END;
00422    4           END;
00423    3       END ADD$RO;
00424    2
00425    1
00426    1       COMPLIMENT: PROCEDURE(NUMB);
00427    1           DECLARE NUMB BYTE;
00428    2           DO CASE NUMB;
00429    2               HOLD=.RO;
00430    2               HOLD=.R1;
00431    3               HOLD=.R2;
00432    3           END;
00433    3           IF SIGN$O(NUMB) THEN SIGNO(NUMB) = NEGITIVE;
00434    2           ELSE SIGNO(NUMB) = POSITIVE;
00435    2           DO CTR=0 TO 9;
00436    2               H$BYTE(CTR)=99H - H$BYTE(CTR);
00437    2           END;
00438    3       END COMPLIMENT;
00439    2
00440    1
00441    1       CHECK$RESULT: PROCEDURE;
00442    1           IF SHR(R2,4)=9 THEN CALL COMPLIMENT(2);
00443    2           IF SHR(R2,4)<>0 THEN OVERFLOW=TRUE;
00444    2       END CHECK$RESULT;
00445    2
00446    1
00447    1       CHECK$SIGN: PROCEDURE;
00448    1           IF SIGNO AND SIGN1 THEN
00449    2           DO;
00450    2               SIGN2=POSITIVE;
00451    2               RETURN;
00452    3           END;
00453    3           SIGN2=NEGITIVE;
00454    2           IF NOT SIGNO AND NOT SIGN1 THEN RETURN;
00455    2           IF SIGNO THEN CALL COMPLIMENT(1);
00456    2           ELSE CALL COMPLIMENT(0);
00457    2       END CHECK$SIGN;
00458    2
00459    1
00460    1       LEADING$ZEROES: PROCEDURE (ADDR) BYTE;
00461    1           DECLARE COUNT BYTE, ADDR ADDRESS;
00462    2           COUNT=0;
00463    2           BASE=ADDR;
00464    2           DO CTR=0 TO 9;
00465    2               IF (B$BYTE(CTR) AND 0F0H) <> 0 THEN RETURN COUNT;
00466    2               COUNT=COUNT + 1;
00467    3               IF (B$BYTE(CTR) AND 0FH) <> 0 THEN RETURN COUNT;
00468    3               COUNT=COUNT + 1;
00469    3           END;
00470    3           RETURN COUNT;
00471    2       END LEADING$ZEROES;
00472    2
00473    1
00474    1       CHECK$DECIMAL: PROCEDURE;
00475    1           IF DEC$PT2<>(CTR:=C$BYTE(3)) THEN
00476    2           DO;
00477    2               BASE=.R2;
00478    2               IF DEC$PT2 > CTR THEN CALL SHIFT$RIGHT(DEC$PT2-CTR);
00479    3               ELSE CALL SHIFT$LEFT(CTR-DEC$PT2);
00480    3           END;
00481    3           IF LEADING$ZEROES(.R2) < 19 - C$BYTE(2) THEN OVERFLOW = TRUE;
00482    2       END CHECK$DECIMAL;
00483    2
00484    1
00485    1       ADD: PROCEDURE;
00486    1           OVERFLOW=FALSE;
00487    2           CALL ALLIGN;
00488    2           CALL CHECK$SIGN;
00489    2           CALL ADD$RO(.R1,.R2);
00490    2           CALL CHECK$RESULT;
00491    2       END ADD;
00492    2
00493    1
00494    1       ADD$SERIES: PROCEDURE(COUNT);
00495    1           DECLARE (I,COUNT) BYTE;
00496    2           DO I=1 TO COUNT;
00497    2               CALL ADD$RO(.R2,.R2);
00498    3           END;
00499    2       END ADD$SERIES;
00500    1
00501    1
```

151

```
00502    1
00503    1          SETSMULTSDIV: PROCEDURE;
00504    2              OVERFLOW=FALSE;
00505    2              IF (SIGNO AND SIGN1) OR
00506    2                  (NOT SIGNO AND NOT SIGN1) THEN SIGN2=POSITIVE;
00507    2              ELSE SIGN2=NEGITIVE;
00508    2              CALL FILL(.R2,10,0);
00509    2          END SETSMULTSDIV;
00510    1
00511    1
00512    1          R1SGREATER: PROCEDURE BYTE;
00513    2              DECLARE I BYTE;
00514    2              DO CTR=0 TO 9;
00515    2                  IF R1(CTR)>(I:=99H-R0(CTR)) THEN RETURN TRUE;
00516    3                  IF R1(CTR)<I THEN RETURN FALSE;
00517    3              END;
00518    2              RETURN TRUE;
00519    2          END R1SGREATER;
00520    1
00521    1
00522    1          MULTIPLY: PROCEDURE(VALUE);
00523    2              DECLARE VALUE BYTE;
00524    2              IF VALUE<>0 THEN CALL ADDSSERIES(VALUE);
00525    2              BASE=.R0;
00526    2              CALL ONESLEFT;
00527    2          END MULTIPLY;
00528    1
00529    1
00530    1          DIVIDE: PROCEDURE;
00531    2              DECLARE (I,J,K,LZO,LZ1) BYTE;
00532    2              CALL SETSMULTSDIV;
00533    2              IF(LZO:=LEADINGSZEROES(BASE:=.R0))<>
00534    2                  (LZ1:=LEADINGSZEROES(.R1)) THEN
00535    2              DO;
00536    2                  IF LZO>LZ1 THEN
00537    3                  DO;
00538    3                      CALL SHIFTSLEFT(I:=LZO-LZ1);
00539    4                      DECSPTO=DECSPTO + I;
00540    4                  END;
00541    3                  ELSE DO;
00542    3                      CALL SHIFTSRIGHT(I:=LZ1-LZO);
00543    4                      DECSPTO=DECPTO -I;
00544    4                  END;
00545    3              END;
00546    2              DECPT2= 20 - LZ1 + DECPTO - DECPT1;
00547    2              CALL COMPLIMENT(0);
00548    2              DO I=LZ1 TO 19;
00549    2                  J=0;
00550    3                  DO WHILE R1SGREATER;
00551    3                      CALL ADDSR0(.R1,.R1);
00552    4                      J=J+1;
00553    4                  END;
00554    3                  K=SHR(I,1);
00555    3                  IF I THEN R2(K)=R2(K) OR J;
00556    3                  ELSE R2(K)=R2(K) OR SHL(J,4);
00557    3              END;
00558    2          END DIVIDE;
00559    1
00560    1
00561    1
00562    1
00563    1          LOADSASCHAR: PROCEDURE(CHAR);
00564    2              DECLARE CHAR BYTE;
00565    2              IF (SWITCH:=NOT SWITCH) THEN
00566    2                  BSBYTE(RSPTR)=BSBYTE(RSPTR) OR SHL(CHAR - 30H,4);
00567    2              ELSE BSBYTE(RSPTR:=RSPTR-1)=CHAR - 30H;
00568    2          END LOADSASCHAR;
00569    1
00570    1
00571    1          LOADSNUMBERS: PROCEDURE(ADDR,CNT);
00572    2              DECLARE ADDR ADDRESS, (I,CNT)BYTE;
00573    2              HOLD=RES(ADDR);
00574    2              CTR=CNT;
00575    2              DO INDEX = 1 TO CNT;
00576    2                  CTR=CTR-1;
00577    3                  CALL LOADSASCHAR(HSBYTE(CTR));
00578    3              END;
00579    2              CALL INCSPTR(5);
00580    2          END LOADSNUMBERS;
00581    1
00582    1
00583    1          SETSLOAD: PROCEDURE (SIGNSIN);
00584    2              DECLARE SIGNSIN BYTE;
00585    2              DO CASE (CTR:=CSBYTE(4));
00586    2                  BASE=.R0;
00587    3                  BASE=.R1;
00588    3                  BASE=.R2;
00589    3              END;
00590    2              DECSPTO(CTR)=CSBYTE(3);
00591    2              SIGNO(CTR)=SIGNSIN;
00592    2              CALL FILL (BASE,10,0);
00593    2              RSPTR=9;
00594    2              SWITCH=FALSE;
00595    2          END SETSLOAD;
00596    1
00597    1
00598    1          LOADSNUMERIC: PROCEDURE;
00599    2              CALL SETSLOAD(1);
00600    2              CALL LOADSNUMBERS(CSADDR,CSBYTE(2));
00601    2          END LOADSNUMERIC;
00602    1
```

152

```
00603   1
CC604   1      LOAD$NUM$LIT: PRCCECURE;
00605   2          CECLARE(LIT$SIZE,FLAG) BYTE;
00606   2
CC607   2          CHAR$SIGN: PRCCEDURE;
00608   3              LIT$SIZE=LIT$SIZE - 1;
00609   3              HCLD=HCLD + 1;
C0610   3          END CHAR$SIGN;
C0611   2
00612   2          LIT$SIZE=C$EYTE(2);
00613   2          HCLD=C$ACCR;
00614   2          IF H$BYTE='-' THEN
00615   2          CC;
00616   2              CALL CHAR$SIGN;
C0617   2              CALL SET$LOAD(NEGITIVE);
00618   3          ENC;
00619   2          ELSE CC;
C0620   2              IF H$BYTE='+' THEN CALL CHAR$SIGN;
00621   3              CALL SET$LOAD(PCSITIVE);
00622   3          ENC;
00623   2          FLAG=0;
C0624   2          CTR=LIT$SIZE;
C0625   2          DO INDEX=1 TC LIT$SIZE;
C0626   2              CTR=CTR-1;
C0627   3              IF H$BYTE(CTR)='.' THEN FLAG=LIT$SIZE - (CTR+1);
C0628   3              ELSE CALL LOAD$A$CHAR(H$BYTE(CTR));
00629   3          ENC;
CC630   2          CEC$PTO(C$BYTE(4))= FLAG;
00631   2          CALL INC$PTR(5);
00632   2      ENC LOAD$NUM$LIT;
00633   1
00634   1
CC635   1      STCRE$CNE: PROCECURE;
00636   1          IF(SWITCH:=NCT SWITCH) THEN
C0637   2              B$BYTE=SHR(H$BYTE,4) CR '0';
CO638   2          ELSE CC;
00639   2              HCLD=HCLD-1;
C0640   3              B$BYTE=(H$BYTE AND 0FH) CR '0';
CO641   3          ENC;
C0642   2          BASE=BASE-1;
00643   2      END STCRE$CNE;
C0644   1
00645   1
C0646   1      STCRE$AS$CHAR: PRCCEDURE(CCUNT);
C0647   2          CECLARE CCUNT BYTE;
C0648   2          SWITCH=FALSE;
C0649   2          HCLD=.R2 + 5;
CO650   2          DC CTR=1 TO CCUNT;
00651   2              CALL STCRE$CNE;
C0652   3          ENC;
C0653   2      END STCRE$AS$CHAR;
CO654   1
CC655   1
00656   1      SET$ZCNE: PRCCECLRE (ACCR);
C0657   2          CECLARE ACCR ADDRESS;
C0658   2          IF NCT SIGN2 THEN
C0659   2          DO;
C0660   2              BASE=ACCR;
00661   3              B$BYTE=B$BYTE CR ZONE;
00662   3          ENC;
00663   2          CALL INC$PTR(4);
00664   2      ENC SET$ZONE;
00665   1
C0666   1
CO667   1      SET$SIGN$SEP: PRCCEDURE (ADDR);
00668   2          CECLARE ADCR ADDRESS;
00669   2          BASE=ACCR;
CC670   2          IF SIGN2 THEN B$BYTE='+';
00671   2          ELSE B$BYTE='-';
00672   2          CALL INC$PTR(4);
00673   2      ENC SET$SIGN$SEP;
C0674   1
CC675   1
C0676   1      STCRE$NUMERIC: PRCCEDURE;
00677   2          CALL CHECK$DECIMAL;
00678   2          BASE=C$ACCR + C$BYTE(2) -1;
00679   2          CALL STCRE$AS$CHAR(C$BYTE(2));
CC680   1      ENC STORE$NUMERIC;
00681   1
0C682   1
0C683   1
C0684   1
C0685   1
00686   1          /* * * * * * * * * * INPUT-OUTPUT ACTIONS * * * * * * * * * * * * */
CC687   1
CC688   1
C0689   1      CECLARE
CC690   1
C0691   1      FLAG$CFFSET              LIT          '33',
C0692   1      EXTENT$GFFSET           LIT          '12',
00693   1      REC$NO                  LIT          '32',
CO694   1      FTR$CFFSET              LIT          '17',
CC695   1      BUFF$LENGTH             LIT          '128',
CC696   1      VAR$END                 LIT          'CR',
C0697   1      TERMINATOR              LIT          '1AH',
C0698   1      END$CF$RECCRC           BYTE,
C0699   1      INVALIC                 BYTE,
C0700   1      RANDCM$FILE             BYTE,
00701   1      CLRRENT$FLAG            BYTE,
00702   1      FCB$BYTE                BASEC CURRENT$FCB          BYTE,
C0703   1      FCB$ACCR                BASEC CURRENT$FCB          ADDRESS,
00704   1      BUFF$FTR                ADDRESS,
CO705   1      BUFF$END                ADDRESS,
C0706   1      BUFF$START              ADDRESS,
CO707   1      BUFF$BYTE               BASED BUFF$PTR        BYTE,
C0708   1      CCN$BUFF                ADDRESS  INITIAL (80H),
CC709   1      CON$BYTE                BASED CON$BUFF          BYTE,
CO710   1      CCN$INPUT               ADDRESS  INITIAL (82H);
.0G711  1
```

153

```
UU712    1           ACCEPT: PRCCECLRE;
C0713    1               CALL CRLF;
C0714    2               CALL PRINTSCHAR(3FH);
C0715    2               CALL CRLF;
C0716    2               CALL FILL(CCNSINPUT,(CONSBYTE:=CSBYTE(2)),' ');
C0717    2               CALL READ(CCNSBUFF);
00718    2               CALL MCVE(CCNSINPLT,RES(CSADDR),CONSBYTE);
00719    2               CALL INCSPTR(3);
00720    2           END ACCEPT;
00721    2
00722    1
00723    1
00724    1           DISPLAY: PRCCECLRE;
00725    2               BASE=CSACDR;
00726    2               CALL CRLF;
00727    2               DC CTR= O TC CSBYTE(2) - 1;
00728    2                   CALL PRINTSCHAR(BSBYTE(CTR));
00729    3               END;
00730    2               CALL INCSPTR(3);
00731    2           END DISPLAY;
00732    1
00733    1
00734    1           SETSFILESTYFE: FRCCEDURE(TYPE);
00735    2               DECLARE TYPE BYTE;
00736    2               BASE=CSADDR;
00737    2               BSBYTE(FLAGSCFFSET)=TYPE;
00738    2           END SETSFILESTYPE;
00739    1
00740    1
00741    1           GETSFILESTYPE: FRCCEDURE BYTE;
00742    2               BASE=CSADDR;
00743    2               RETURN BSBYTE(FLAGSCFFSET);
00744    2           END GETSFILESTYFE;
00745    1
00746    1
00747    1           SETSISC: PRCCECLRE;
00748    2               ENDSCFSRECCRC,INVALID=FALSE;
00749    2               IF CSACCF=CLRRENTSFCB THEN RETURN;
00750    2               /* STORE CLRRENT PCINTERS AND SET INTERNAL WRITE MARK */
00751    2               BASE=CURRENTSFCB;
00752    2               FCBSADDR(PTRSCFFSET)=BUFFSPTR;
00753    2               FCBSBYTE(FLAGSCFFSET)=CURRENTSFLAG;
00754    2               /* LOAC NEW VALUES */
00755    2               BLFFSENC=(BLFFSSTART:=(CURRENTSFCB:=CSADDR)+STARTSCFFSET)
00756    2                   + BLFFSLENGTH;
00757    2               CLRRENTSFLAG=FCBSBYTE(FLAGSCFFSET);
00758    2               BLFFSPTR=FCBSADDR(PTRSOFFSET);
00759    2           END SETSISC;
00760    1
00761    1
00762    1           CPENSFILE: PRCCECURE(TYPE);
00763    2               DECLARE TYPE BYTE;
00764    2               CALL SETSFILESTYPE(TYPE);
00765    2               CTR=OPEN(CLRRENTSFCB:=CSADDR);
00766    2               DC CASE TYPE-1;
00767    2                   /* INFLT */
00768    2                   DC;
00769    3                       IF CTR=255 THEN CALL PRINTSERROR('NF');
00770    4                       FCBSADDR(PTRSDFFSET)=CURRENTSFCB+100H;
00771    4                   ENC;
00772    3                   /* CLTFLT */
00773    3                   DC;
00774    3                       CALL CELETE;
00775    3                       CALL MAKE(CSADDR);
00776    4                       FCBSADCR(PTRSUFFSET)=CURRENTSFCB+STARTSDFFSET-1;
00777    4                   ENC;
00778    3                   /* I-C */
00779    3                   DC;
00780    3                       IF CTR=255 THEN CALL FATALSERRCR('NF');
00781    4                       FCBSADDR(PTRSGFFSET)=CURRENTSFCB + 100H;
00782    4                   ENC;
00783    3               END;
00784    2               CLRRENTSFCB=C;      /* FORCE A PARAMETER LDAC */
00785    2               CALL SETSISC;
00786    2               CALL INCSPTR(2);
00787    2           END CPENSFILE;
00788    1
00789    1
00790    1           WRITESMARK: FRCCECURE BYTE;
00791    2               RETURN RCL(CURRENTSFLAG,1);
00792    2           END WRITESMARK;
00793    1
00794    1
00795    1           SETSWRITESMARK: FRCCEDURE;
00796    2               CLRRENTSFLAG=CUFRENTSFLAG CR 80H;
00797    2           END SETSWRITESMARK;
00798    1
00799    1
00800    1           WRITESRECORC: FRCCEDURE;
00801    2               IF NOT SFR(CURRENTSFLAG,1) THEN CALL FATALSERRDR('WI');
00802    2               CALL SETSCMA;
00803    2               CLRRENTSFLAG=CURRENTSFLAG AND JFH;
00804    2               IF (CTR:=DISKSWRITE) =0 THEN RETUFN;
00805    2               INVALIC=TRLE;
00806    2           END WRITESRECCRC;
00807    1
00808    1
00809    1           READSRECCRC: FRCCECURE;
00810    2               CALL SETSCMA;
00811    2               IF WRITESMAFK THEN CALL WRITESRECORD;
00812    2               IF (CTR:=DISKSREAD)=0 THEN RETURN;
00813    2               IF CTR=1 THEN ENDSOFSRECORD=TRUE;
00814    2               ELSE INVALIC=TRUE;
00815    2           END READSRECCRC;
00816    1
```

154

```
00817   1
00818   1        READ$BYTE: PROCEDURE BYTE;
00819   2           IF (BUFF$PTR:=BUFF$PTR + 1) >= BUFF$END THEN
00820   2           DO;
00821   2               CALL READ$RECORD;
00822   3               IF END$CF$RECORD THEN RETURN TERMINATOR;
00823   3               BUFF$PTR=BUFF$START;
00824   3           END;
00825   2           RETURN ELFF$BYTE;
00826   2        END READ$BYTE;
00827   1
00828   1
00829   1        WRITE$BYTE: PROCEDURE (CHAR);
00830   2           DECLARE CHAR BYTE;
00831   2           IF (BUFF$PTR:=BUFF$PTR+1) >= BUFF$END THEN
00832   2           DO;
00833   2               CALL WRITE$RECORD;
00834   3               BUFF$PTR=BUFF$START;
00835   3           END;
00836   2           CALL SET$WRITE$MARK;
00837   2           BUFF$BYTE=CHAR;
00838   2        END WRITE$BYTE;
00839   1
00840   1
00841   1        WRITE$END$MARK: PROCEDURE;
00842   2           CALL WRITE$BYTE(CR);
00843   2           CALL WRITE$BYTE(LF);
00844   2        END WRITE$END$MARK;
00845   1
00846   1
00847   1        READ$END$MARK: PROCEDURE;
00848   2           IF READ$BYTE<>CR THEN CALL PRINT$ERROR('EM');
00849   2           IF READ$BYTE<>LF THEN CALL PRINT$ERROR('EM');
00850   2        END READ$END$MARK;
00851   1
00852   1
00853   1        READ$VARIABLE: PROCEDURE;
00854   2           CALL SET$I$C;
00855   2           BASE=C$ADDR(1);
00856   2           DO ASCTR=0 TC C$ADDR(2)-1;
00857   2               IF (CTR:=(B$BYTE(ASCTR):=READ$BYTE)) = VAR$END THEN
00858   3               DO;
00859   3                   CTR=READ$BYTE;
00860   4                   RETURN;
00861   4               END;
00862   3               IF CTR=TERMINATOR THEN
00863   3               DO;
00864   3                   END$OF$RECORD=TRUE;
00865   4                   RETURN;
00866   4               END;
00867   3           END;
00868   2           CALL READ$END$MARK;
00869   2        END READ$VARIABLE;
00870   1
00871   1
00872   1        WRITE$VARIABLE: PROCEDURE;
00873   2           DECLARE CCUNT ADDRESS;
00874   2           CALL SET$I$C;
00875   2           BASE=C$ADDR(1);
00876   2           CCUNT=C$ADDR(2);
00877   2           DO WHILE(B$BYTE(CCUNT:=COUNT-1)<>' ')AND (CCUNT<>0);
00878   2           END;
00879   2           DO ASCTR=0 TC CCUNT;
00880   2               CALL WRITE$BYTE(B$BYTE(ASCTR));
00881   3           END;
00882   2           CALL WRITE$END$MARK;
00883   2        END WRITE$VARIABLE;
00884   1
00885   1
00886   1        READ$TO$MEMORY: PROCEDURE;
00887   2           CALL SET$I$C;
00888   2           BASE=C$ADDR(1);
00889   2           DO ASCTR=0 TC C$ADDR(2)-1;
00890   2               IF (B$BYTE(ASCTR):=READ$BYTE)=TERMINATOR THEN
00891   3               DO;
00892   3                   END$OF$RECORD=TRUE;
00893   4                   RETURN;
00894   4               END;
00895   3           END;
00896   2           CALL READ$END$MARK;
00897   2        END READ$TO$MEMORY;
00898   1
00899   1
00900   1        WRITE$FROM$MEMORY: PROCEDURE;
00901   2           CALL SET$I$C;
00902   2           BASE=C$ADDR(1);
00903   2           DO ASCTR=0 TC C$ADDR(2)-1;
00904   2               CALL WRITE$BYTE(B$BYTE(ASCTR));
00905   3           END;
00906   2           CALL WRITE$END$MARK;
00907   2        END WRITE$FROM$MEMORY;
00908   1
```

```
C0909      1          /* * * * * * * * * RANDOM I-O PROCEDURES * * * * * * * * */
(0910      1
C0911      1
C0912      1
CC913      1       SET$RANDOM$POINTER: PROCEDURE;
C0914      2          /*
CO915      2          THIS PROCEDURE READS THE RANDOM KEY AND COMPUTES
00916      2          WHICH RECORD NEEDS TO BE AVAILABLE IN THE BUFFER
00917      2          THAT RECORD IS MADE AVAILABLE AND THE POINTERS
00918      2          SET FOR INPUT OR OUTPUT
C0919      2          */
CO920      2          DECLARE (BYTE$COUNT,RECORD) ADDRESS,
C0921      2              EXTENT BYTE;
30922      2          CALL SET$IS$C;
C0923      2          BYTE$COUNT=(C$ADDR(2)+2)*CONVERT$TO$HEX(C$ADDR(3),C$BYTE(8));
00924      2          RECORD=SHR(BYTE$COUNT,7);
00925      2          EXTENT=SHR(RECORD,7);
00926      2          IF EXTENT<>FCB$BYTE(EXTENT$OFFSET) THEN
00927      2          DO;
CC928      3          IF WRITE$MARK THEN CALL WRITE$RECORD;
C0929      3          CALL CLOSE(C$ADDR);
CO930      3              FCB$BYTE(EXTENT$OFFSET)=EXTENT;
00931      3              IF OPEN(C$ADDR)<>0 THEN
00932      3              DO;
00933      3                  IF SHR(CURRENT$FLAG,1) THEN CALL MAKE(C$ADDR);
00934      4                  ELSE INVALID=TRUE;
CC935      4              END;
00936      3          END;
0C937      2          BUFF$PTR=(BYTE$COUNT AND 7FH) + BUFF$START -1;
CO938      2          IF FCB$BYTE(REC$NO)<>(CTR:=LOW(RECORD)AND 7FH) THEN
00939      2          DO;
C0940      2          FCB$BYTE(32)=CTR;
00941      3              CALL READ$RECORD;
00942      3          END;
00943      2       END SET$RANDOM$POINTER;
0C944      1
C0945      1
CC946      1       GET$REC$NUMBER: PROCEDURE;
00947      2          DECLARE (RECNUM, K) ADDRESS,
CC948      2              (I,CNT) BYTE,
CC949      2              J(4) ADDRESS INITIAL (10000,1000,100,10),
CC950      2              BUFF(5) BYTE;
C0951      2
C0952      2          REC$NUM=SHL(FCB$BYTE(EXTENT$OFFSET),7)+FCB$BYTE(REC$NO);
00953      2          DO I=0 TO 3;
00954      2          CNT=0;
00955      3              DO WHILE REC$NUM>=(K:=J(I));
00956      3                  REC$NUM=REC$NUM - K;
00957      4                  CNT=CNT + 1;
0C958      4              END;
CC959      3              BUFF(I)=CNT + '0';
CC960      3          END;
0C961      2          BUFF(4)=REC$NUM+'0';
00962      2          IF (I:=C$BYTE(8))<=5 THEN
0C963      2          CALL MOVE(.BUFF+4-I,C$ADDR(3),I);
00964      2          ELSE DO;
CC965      2          CALL FILL(C$ADDR,I-5,' ');
00966      2              CALL MOVE(.BUFF,C$ADDR(3)+I-6, 5);
CC967      3          END;
CC968      2       END GET$REC$NUMBER;
00969      1
CC97C      1
00971      1       WRITE$ZERO$RECCRD: PROCEDURE;
C0972      2          DO A$CTR=1 TO C$ADDR(2);
CC973      2              CALL WRITE$BYTE(0);
0C974      3          END;
00975      2       END WRITE$ZERO$RECORD;
00976      1
00977      1
00978      1       WRITE$RANDOM: PROCEDURE;
00979      2          CALL SET$RANDOM$POINTER;
(C980      2          CALL WRITE$FROM$MEMORY;
CC981      2          CALL INC$PTR(5);
CC982      2       END WRITE$RANDOM;
CC983      1
CC984      1
CC985      1       BACK$ONE$RECORD: PROCEDURE;
CC986      2          CALL SET$IS$C;
0C987      2       IF (BUFF$PTR:=BUFF$PTR-(C$ADDR(2)+2))>=BUFF$START THEN RETURN;
0C988      2          BUFF$PTR=BUFF$END-(BUFF$START - BUFF$PTR);
C0989      2          IF (FCB$BYTE(REC$NO):=FCB$BYTE(REC$NO)-1)=255 THEN
CC990      2          DO;
CC991      2          FCB$BYTE(EXTENT$OFFSET)=FCB$BYTE(EXTENT$OFFSET)-1;
0C992      3              IF OPEN(C$ADDR)<> 0 THEN
CC993      3              DO;
0C994      3              CALL PRINT$ERROR('OP');
0C995      4                  INVALID=TRUE;
CC996      4              END;
0C997      3              FCB$BYTE(REC$NO)=127;
CC998      3          END;
CC999      3          CALL READ$RECORD;
C1COO      2       END BACK$ONE$RECORD;
01C01      1
01C02      1
C1C03      1          /* * * * * * * * * * * * * MOVES * * * * * * * * * * * * * * */
C1C04      1
C1C05      1
C1C06      1       INC$FCLD: PROCEDURE;
01CC7      2          FCLD=FCLD + 1;
C1C08      2          CTR=CTR + 1;
C1C09      2       END INC$FCLC;
C1C10      1
01C11      1
C1C12      1       LOAD$INC: PROCEDURE;
01C13      2          FS$BYTE=B$BYTE;
J1C14      2          BASE=BASE+1;
C1C15      2          CALL INC$FCLC;
C1C16      2       END LOAD$INC;
01C17      1
```

156

```
C1018   1
C1019   1        CHECKSEDIT: PROCEDURE(CHAR);
01020   2            DECLARE CHAR BYTE;
01021   2            IF (CHAR='0') OR (CHAR='/') THEN CALL INCSHOLD;
01022   2            ELSE IF CHAR='B' THEN
01023   2            DO;
01024   2                HSBYTE=' ';
01025   3                CALL INCSHOLD;
C1026   3            END;
C1027   2            ELSE IF CHAR='A' THEN
C1028   2            DO;
01029   2                IF NCT LETTER(BSBYTE) THEN CALL PRINTSERROR('IC');
C1030   3                CALL LOADSINC;
01031   3            END;
01032   2            ELSE IF CHAR='9' THEN
01033   2            DO;
01034   2                IF NOT NUMERIC (BSBYTE) THEN CALL PRINTSERROR('1C');
C1035   3                CALL LOADSINC;
C1036   3            END;
01037   2            ELSE CALL LOADSINC;
C1038   2        END CHECKSEDIT;
C1039   1
C1040   1            /* * * * * * * * * * * MACHINE ACTIONS * *  * * * * * * * * * */
01041   1
01042   1
01043   1        STOP: PROCEDURE;
01044   2            CALL PRINT(.'EOF    $');
01045   2            GO TO BOOT;
C1046   2        END STOP;
01047   1
01048   1
C1049   1            /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C1050   1
01051   1                  THE PROCEDURE BELOW CONTROLS THE EXECUTION OF THE CODE.
01052   1                  IT DECODES EACH OP-CODE AND PERFORMS THE ACTIONS
01053   1
01054   1            * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
01055   1
C1056   1
01057   1        EXECUTE: PROCEDURE;
C1058   2            DO FOREVER;
C1059   2                DO CASE GETSOPSCODE;
01060   3
01061   3                    ;       /* CASE ZERO NOT USED */
01062   4
C1063   4        /* ACC */
C1064   4
01065   4                        CALL ADD;
C1066   4
01067   4        /* SUB */
C1068   4
01069   4                        DO;
01070   4                            CALL COMPLIMENT(0);
01071   5                            IF SIGNO THEN SIGNO=NEGITIVE;
01072   5                            ELSE SIGNO=POSITIVE;
01073   5                            CALL ADD;
C1074   5                        END;
C1075   4
C1076   4        /* MUL */
01077   4
C1078   4                        DO;
C1079   4                            DECLARE I BYTE;
C1080   5                            CALL SETSMULTSDIV;
01081   5                            DECPT1,DECPT2=DECPT1 + DECPT0;
01082   5                            CALL ALLIGN;
01083   5                            CALL MULTIPLY(SHR(R1(I:=9),4));
01084   5                            DO INDEX=1 TO 9;
01085   5                                CALL MULTIPLY(R1(I:=I-1) AND 0FH);
C1086   6                                CALL MULTIPLY(SHR(R1(I),4));
C1087   6                            END;
C1088   5                        END;
C1089   4
01090   4        /* DIV */
01091   4
01092   4                        CALL DIVIDE;
C1093   4        /* NEG */
C1094   4
01095   4                        BRANCHSFLAG=FALSE;
01096   4
01097   4        /* STP */
01098   4
C1099   4                        CALL STOP;
C1100   4
01101   4        /* STI */
01102   4
01103   4                        CALL STORESIMMEDIATE;
01104   4
01105   4
C1106   4        /* RND */
01107   4
C1108   4                        DO;
C1109   4                            CALL STORESIMMEDIATE;
C1110   5                            CALL FILL(.R2,10,0);
01111   5                            R2(9)=1;
01112   5                            CALL ADD;
C1113   5                        END;
C1114   4
01115   4        /* RET */
C1116   4
01117   4                        DO;
C1118   4                            IF CSADDR<>0 THEN
C1119   5                            DO;
C1120   5                                ASCTR=CSADDR;
01121   6                                CSADDR=0;
01122   6                                PROGRAMSCOUNTER=ASCTR;
C1123   6                            END;
01124   6                            ELSE CALL INCSPTR(2);
C1125   5                        END;
```

157

```
C1126   4       /* CLS */
01127   4
C1128   4
C1129   4               CC;
01130   4                   CALL SET$IS0;
01131   5                   IF WRITE$MARK THEN CALL WRITE$RECORD;
01132   5                   CALL CLOSE(CSADDR);
01133   5                   CALL INC$PTR(2);
01134   5               END;
01135   4
C1136   4       /* SER */
01137   4
C1138   4               CC;
01139   4                   IF OVERFLOW   THEN PROGRAM$CCUNTER = CSADDR;
C1140   5                   ELSE CALL INC$PTR(2);
01141   5               END;
01142   4       /* BFN */
01143   4
01144   4               FRCGRAM$CCUNTER=CSADDR;
C1145   4
01146   4       /* CFN */
01147   4
01148   4               CALL OPEN$FILE(1);
C1149   4
C1150   4       /* CP1 */
01151   4
01152   4               CALL OPEN$FILE(2);
01153   4
C1154   4       /* CF2 */
01155   4
01156   4               CALL OPEN$FILE(3);
C1157   4
C1158   4       /* RGT */
C1159   4
C1160   4               CC;
01161   4                   IF NOT SIGN2 THEN
01162   5                       BRANCH$FLAG=NCT BRANCH$FLAG;
01163   5                   CALL CONDITICNAL$BRANCH(0);
C1164   5               END;
01165   4
01166   4       /* RLT */
01167   4
01168   4               CC;
C1169   4                   IF SIGN2 THEN
C1170   5                       BRANCH$FLAG=NCT BRANCH$FLAG;
01171   5                   CALL CONDITICNAL$BRANCH(0);
01172   5               END;
C)173   4
01174   4       /* REC */
01175   4
C1176   4               CC;
01177   4                   IF NOT ZERO$RESULT THEN
C1178   5                       BRANCH$FLAG=NCT BRANCH$FLAG;
C1179   5                   CALL CONDITICNAL$BRANCH(0);
C1180   5               END;
C1181   4
C1182   4       /* INV */     .
C1183   4
01184   4               CALL INCREMENT$OR$BRANCH(INVALID);
C1185   4
01186   4       /* EOR */
01187   4
C1188   4               CALL INCREMENT$OR$BRANCH(ENC$OF$RECORD);
C1189   4
C1190   4       /* ACC */
01191   4
01192   4               CALL ACCEPT;
01193   4
C1194   4       /* DIS */
C1195   4
01196   4               CALL DISPLAY;
01197   4
C1198   4       /* STC */
C1199   4
01200   4               CC;
01201   4                   CALL DISPLAY;         .
01202   5                   CALL STCP;
01203   5               END;
C1204   4
01205   4       /* LCI */
01206   4
C1207   4               CC;
C1208   4                   CSADDR(3)=CCNVERT$TO$HEX(CSADDR,C$BYTE(2));
C1209   4                   CALL INC$PTR(3);
C1210   5               END;
C1211   4
01212   4       /* CEC */
01213   4
01214   4               CC;
01215   4                   IF CSADDR<>0 THEN CSADDR=CSADDR-1;
01216   4                   IF CSADDR=0 THEN PROGRAM$CCUNTER=CSADDR(1);
C1217   5                   ELSE CALL INC$PTR(4);
C1218   5               END;                      .
C1219   4
01220   4       /* STC */
01221   4
01222   4               CC;
01223   4                   CALL STORE$NUMERIC;
01224   5                   CALL INC$PTR(4);
01225   5               END;
C1226   4
C1227   4       /* ST1 */
C1228   4
C1229   4               CC;
01230   4                   CALL STCRE$NUMERIC;
01231   5                   CALL SET$ZONE(CSADDR+C$BYTE(2)-1);
_C1232  5               END;
```

```
01233   4     /* ST2 */
01234   4
01235   4
01236   4              DC;
01237   4                      CALL STORE$NUMERIC;
01238   5                      CALL SET$ZONE(CSADCR);
C1239   5              END;
01240   4
01241   4     /* ST3 */
01242   4
01243   4              DC;
C1244   4                      CALL CHECK$DECIMAL;
01245   4                      BASE=CSADDR + CSBYTE(2) - 1;
01246   5                      CALL STORE$AS$CHAR(CSBYTE(2) - 1);
C1247   5                      CALL SET$SIGN$SEP(CSADCR + CSBYTE(2) -1);
C1248   5              END;
C1249   4
C1250   4     /* ST4 */
01251   4
01252   4              CC;
01253   4                      CALL CHECK$DECIMAL;
01254   4                      BASE=CSADDR + CSBYTE(2);
01255   5                      CALL STORE$AS$CHAR(CSBYTE(2)-1);
01256   5                      CALL SET$SIGN$SEP(CSADCR);
01257   5              END;
C1258   4
C1259   4     /* ST5 */
C1260   4
C1261   4              CC;
01262   4                      CALL CHECK$DECIMAL;
01263   5                      R2(9)=R2(9) CR SIGN2;
01264   5                      CALL MOVE(.R2 + 9 - CSBYTE(2),CSADDR,CSBYTE(2));
01265   5                      CALL INCSPTR(4);
01266   5              END;
01267   4
C1268   4     /* LCD */
C1269   4
C1270   4              CALL LOAD$NUM$LIT;
C1271   4
01272   4     /* LC1 */
C1273   4
01274   4              CALL LCAD$NUMERIC;
01275   4
C1276   4     /* LC2 */
01277   4
C1278   4              CC;
C1279   4                      DECLARE I BYTE;
C1280   5                      HCLD=CSADDR;
01281   5                      IF CHECK$FOR$SIGN(CTR:=HSBYTE(I:=CSBYTE(2)-1)) THEN
01282   5                      DC;
01283   5                              CALL SET$LCAD(POSITIVE);
01284   6                              I=I+1;
01285   6                      END;
01286   5                      ELSE DO;
01287   5                              CALL SET$LCAD(NEGITIVE);
C1288   6                              CALL LOAD$AS$CHAR(CTR-ZONE);
C1289   6                      END;
C1290   5                      CALL LOAD$NUMBERS(CSADDR,I);
01291   5              END;
C1292   4
C1293   4     /* LC3 */
01294   4
01295   4              CC;
01296   4                      HCLD=CSADDR;
C1297   5                      IF CHECK$FOR$SIGN(HSBYTE) THEN
C1298   5                      DC;
C1299   5                              CALL SET$LCAD(POSITIVE);
C1300   6                              CALL LCAD$NUMBERS(CSADDR,CSBYTE(2));
01301   6                      END;
01302   5                      ELSE DO;
01303   5                              CALL SET$LCAD(NEGITIVE);
C1304   6                              CALL LOAD$NUMBERS(CSADCR+1,CSBYTE(2)-1);
01305   6                              CALL LOAD$AS$CHAR(HSBYTE-ZONE);
01306   6                      END;
01307   5              END;
C1308   4
C1309   4     /* LC4 */
C1310   4
01311   4              CC;
01312   4                      HOLD=CSADDR;
01313   4                      IF HSBYTE(CSBYTE(2) - 1) = '+' THEN
01314   5                              CALL SET$LOAD(1);
01315   5                      ELSE CALL SET$LOAD(0);
01316   5                      CALL LOAD$NUMBERS(CSADCR,CSBYTE(2) -1);
C1317   5              END;
01318   4
C1319   4     /* LC5 */
01320   4
01321   4              CC;
01322   4                      HOLD=CSADDR;
01323   5                      IF(HSBYTE='+') THEN CALL SET$LOAD(1);
01324   5                      ELSE CALL SET$LOAD(0);
01325   5                      CALL LOAD$NUMBERS(CSADCR,CSBYTE(2)-1);
01326   5              END;
01327   4
C1328   4     /* LC6 */
01329   4
C1330   4              CC;
01331   4                      CECLARE I BYTE;
01332   5                      HOLD=CSADDR;
01333   5                      CALL SET$LCAD(HSBYTE(I:=CSBYTE(2)-1));
C1334   5                      BASE=BASE + 9 - I;
01335   5                      DC CTR = 0 TO I;
01336   5                              BSBYTE(CTR)=HSBYTE(CTR);
01337   6                      END;
C1338   5                      BSBYTE(CTR)=BSBYTE(CTR) AND OFOH;
C1339   5                      CALL INCSPTR(5);
C1340   5              FNC;
```

159

```
01341   4        /* PER */
01342   4
01343   4
01344   4                DC;
01345   4                    BASE=C$ADDR(1)+1;
01346   5                    B$ADDR=C$ADDR(2);
01347   5                    PROGRAM$COUNTER=C$ADDR;
01348   5                END;
01349   4
01350   4        /* CAL */
01351   4
01352   4                CALL C$CMP$NUM$UNSIGNED;
01353   4
01354   4        /* CNS */
01355   4
01356   4                CALL C$CMP$NUM$SIGN;
01357   4
01358   4        /* CAL */
01359   4
01360   4                CALL C$CMP$ALPHA;
01361   4
01362   4        /* RBS */
01363   4
01364   4                DC;
01365   4                CALL BACK$CNE$RECORD;
01366   4                    CALL WRITE$FRCM$MEMORY;
01367   5                    CALL INC$PTR(6);
01368   5                END;
01369   4
01370   4        /* CLS */
01371   4
01372   4                DC;
01373   4                CALL BACK$CNE$RECORD;
01374   5                    CALL WRITE$ZERO$RECORD;
01375   5                    CALL INC$PTR(6);
01376   5                END;
01377   4
01378   4        /* RCF */
01379   4
01380   4                DC;
01381   4                    CALL READ$TO$MEMORY;
01382   5                    CALL INC$PTR(6);
01383   5                END;
01384   4
01385   4        /* WTF */
01386   4
01387   4                DC;
01388   4                    CALL WRITE$FRCM$MEMORY;
01389   4                    CALL INC$PTR(6);
01390   5                END;
01391   4
01392   4        /* RVL */
01393   4
01394   4                CALL READ$VARIABLE;
01395   4
01396   4        /* WVL */
01397   4
01398   4                CALL WRITE$VARIABLE;
01399   4
01400   4        /* SCR */
01401   4
01402   4                DC;
01403   4                    SUBSCRIPT(C$BYTE(2))=
01404   5                        CONVERT$TO$HEX(C$ADDR,C$BYTE(3));
01405   5                    CALL INC$PTR(4);
01406   5                END;
014C7   4
014C8   4        /* SGT */
01409   4
01410   4                CALL STRING$CCMPARE(1);
01411   4
01412   4        /* SLT */
01413   4
01414   4                CALL STRING$CCMPARE(0);
01415   4
01416   4        /* SEC */
01417   4
01418   4                CALL STRING$COMPARE(2);
01419   4
01420   4        /* MCV */
01421   4
01422   4                DC;
01423   4                    CALL MOVE(RES(C$ADDR(1)),RES(C$ADDR),C$ADDR(2));
01424   4                    IF C$ADDR(3)<>0 THEN CALL
01425   5                        FILL(RES(C$ADDR(1)) + C$ADDR(2),C$ADDR(3),' ');
01426   5                    CALL INC$PTR(8);
01427   5                END;
01428   4
01429   4        /* RRS */
01430   4
01431   4                DC;
01432   4                    CALL READ$TO$MEMORY;
01433   5                    CALL GET$REC$NUMBER;
01434   5                    CALL INC$PTR(9);
01435   5                END;
01436   4
01437   4        /* WRS */
01438   4
01439   4                DC;
01440   4                CALL WRITE$FROM$MEMORY;
01441   5                    CALL GET$REC$NUMBER;
01442   5                    CALL INC$PTR(9);
01443   5                END;
```

160

```
00001    1
00002    1              /* THIS PROGRAM TAKES THE CODE OUTPUT FROM THE COBOL COMPILER
00003    1              AND BUILDS THE ENVIRONMENT FOR THE COBOL INTERPRETER */
00004    1
00005    1    10CH:          /* LOAD PCINT */
00006    1
00007    1    DECLARE
00008    1
00009    1    LIT            LITERALLY      'LITERALLY',
00010    1    BDOT           LIT            '0',
00011    1    BDCS           LIT            '5',
00012    1    TRUE           LIT            '1',
00013    1    FALSE          LIT            '0',
00014    1    FOREVER        LIT            'WHILE TRUE',
00015    1    FCB            ADDRESS        INITIAL (5CH),
00016    1    FCB$BYTE       BASED     FCB  BYTE,
00017    1    I              BYTE,
00018    1    ADDR           ADDRESS        INITIAL (100H),
00019    1    CHAR           BASED     ADDR BYTE,
00020    1    BUFF$END       LIT            '100H',
00021    1    INTERP$FCB     (23)      BYTE INITIAL(0,'CINTERP COM',0,0,C,C),
00022    1    CODE$NOT$SET   BYTE           INITIAL (TRUE),
00023    1    READER$LOCATION     LIT       '1C80H',
00024    1    INTERP$ADDRESS      ADDRESS   INITIAL(2000H),
00025    1    INTERP$CONTENT      BASED     INTERP$ADDRESS ADDRESS,
00026    1    I$BYTE         BASED               INTERP$ADDRESS BYTE,
00027    1    CODE$CTR       ADDRESS,
00028    1    C$BYTE         BASED     CODE$CTR BYTE,
00029    1    BASE           ADDRESS,
00030    1    B$ADDR      -  BASED BASE     ADDRESS,
00031    1    B$BYTE         BASED          BASE BYTE;
00032    1
00033    1    MON1: PROCEDURE (F,A);
00034    2         DECLARE F BYTE, A ADDRESS;
00035    2         GO TO BDCS;
00036    2    END MON1;
00037    1
00038    1
00039    1    MON2: PROCEDURE (F,A) BYTE;
00040    2         DECLARE F BYTE, A ADDRESS;
00041    2         GO TO BDCS;
00042    2    END MON2;
00043    1
00044    1
00045    1    PRINT$CHAR: PROCEDURE(CHAR);
00046    2         DECLARE CHAR BYTE;
00047    2         CALL MON1(2,CHAR);
00048    2    END PRINT$CHAR;
00049    1
00050    1
00051    1    CRLF: PROCEDURE;
00052    2         CALL PRINT$CHAR(13);
00053    2         CALL PRINT$CHAR(10);
00054    2    END CRLF;
00055    1
00056    1
00057    1    PRINT: PROCEDURE(A);
00058    2         DECLARE A ADDRESS;
00059    2         CALL CRLF;
00060    2         CALL MON1(9,A);
00061    2    END PRINT;
00062    1
00063    1
00064    1    OPEN: PROCEDURE (A) BYTE;
00065    2         DECLARE A ADDRESS;
00066    2         RETURN MON2(15,A);
00067    2    END OPEN;
00068    1
00069    1
00070    1    MOVE: PROCEDURE(FROM, DEST, COUNT);
00071    2         DECLARE (FROM, DEST, COUNT) ADDRESS,
00072    2         (F BASED FROM, D BASED DEST) BYTE;
00073    2         DO WHILE(COUNT:=COUNT-1)<>0FFFFH;
00074    2              D=F;
00075    3              FROM=FROM+1;
00076    3              DEST=DEST+1;
00077    3         END;
00078    2    END MOVE;
00079    1
00080    1
00081    1    GET$CHAR: PROCEDURE BYTE;
00082    2         IF (ADDR:=ADDR + 1)>=BUFF$END THEN
00083    2         DO;
00084    2              IF MON2(20,FCB)<>0 THEN
00085    3              DO;
00086    3                   CALL PRINT(.'END OF INPUT    $');
00087    4                   GO TO BDOT;
00088    4              END;
00089    3              ADDR=8CH;
00090    3         END;
00091    2         RETURN CHAR;
00092    2    END GET$CHAR;
00093    1
```

161

```
CCC94   1    ` NEXTSCHAR: PROCECURE;
CCC95   1        CHAR=GETSCHAR;
OCC96   2    END NEXTSCHAR;
OCC97   2
CCC58   1
CCC99   1
CO100   1    STCRE: PROCECURE(CCUNT);
CO101   2        DECLARE CCLNT BYTE;
CO102   2        IF CODE$NCT$SET THEN
CO103   2        DO;
CO104   2            CALL FRINT(.'CODE ERRORS');
CO105   3            CALL NEXTSCHAR;
CO106   3            RETLRN;
CO107   3        END;
CO108   2        CO I=1 TC CCLNT;
CO109   2            CSBYTE=CHAR;
CO110   3            CALL NEXTSCHAR;
O0111   3            COCESCTR=CODESCTR+1;
O0112   3         END;
CO113   2    END STCRE;
O0114   1
CO115   1
CO116   1    BACK$STUFF: FROCECURE;
CO117   2        DECLARE (FCLC,STUFF) ADDRESS;
O0118   2        BASE=.HCLC;
CO119   2        CO I=0 TC 3;
CO120   2            BSBYTE(I)=GETSCHAR;
O0121   3        END;
O0122   2        CO FOREVER;
O0123   2            BASE=FCLD;
O0124   3            HDLC=BSADDR;
O0125   3            BSADCR=STUFF;
O0126   3            IF FCLC=0 THEN
O0127   3            DO;
CO128   3                CALL NEXTSCHAR;
CO129   4                RETLRN;
CO130   4            END;
O0131   3        END;
O0132   2    END BACKSSTLFF;
O0133   1
O0134   1
CO135   1    STARTSCODE: FROCECURE;
CO136   2        CODE$NOT$SET=FALSE;
CO137   2        ISBYTE=GETSCHAR;
CO138   2        ISBYTE(I)=GETSCHAR;
CO139   2        COCESCTR=INTERPSCCNTENT;
O0140   2        CALL NEXTSCHAR;
O0141   2    END STARTSCCDE;
O0142   1
O0143   1
O0144   1    GCSCEPENCING: PROCECURE;
CO145   2        CALL STCRE(1);
CO146   2        CALL STCRE(SHL(CHAR,1) + 4);
O0147   2    END GCSCEPENCING;
CC148   1
CO149   1
CC150   1    INITIALIZE: FROCECURE;
O0151   2        DECLARE (CCLNT,WHERE,HCWSMANY) ADDRESS;
O0152   2        BASE=.WHERE;
CO153   2        CO I=0 TC 3;
O0154   2            BSBYTE(I)=GETSCHAR;
O0155   3        END;
CO156   2        BASE=WHERE - 1;
O0157   2        CO CDUNT = 1 TO HCWSMANY;
O0158   2            BSBYTE(CCUNT)=GETSCHAR;
CO159   3        END;
CO160   2        CALL NEXTSCHAR;
O0161   2    END INITIALIZE;
CO162   1
```

162

```
00163   1       BUILD: PROCECURE;
00164   1           DECLARE
00165   2           F2      LIT     '8',
00166   2           F3      LIT     '9',
00167   2           F4      LIT     '21',
00168   2           F5      LIT     '25',
00169   2           F6      LIT     '32',
00170   2           F7      LIT     '36',
00171   2           F9      LIT     '49',
00172   2           F10     LIT     '54',
00173   2           F11     LIT     '6C',
00174   2           F13     LIT     '61',
00175   2           GDP     LIT     '62',
00176   2           INT     LIT     '63',
00177   2           BST     LIT     '64',
00178   2           TER     LIT     '65',
00179   2           SCD     LIT     '66';
00180   2
00181   2
00182   2           DO FOREVER;
00183   3               IF CHAR < F2 THEN CALL STORE(1);
00184   3               ELSE IF CHAR < F3 THEN CALL STORE(2);
00185   3               ELSE IF CHAR < F4 THEN CALL STORE(3);
00186   3               ELSE IF CHAR < F5 THEN CALL STORE(4);
00187   3               ELSE IF CHAR < F6 THEN CALL STORE(5);
00188   3               ELSE IF CHAR < F7 THEN CALL STORE(6);
00189   3               ELSE IF CHAR < F9 THEN CALL STORE(7);
00190   3               ELSE IF CHAR < F10 THEN CALL STORE(9);
00191   3               ELSE IF CHAR < F11 THEN CALL STORE(10);
00192   3               ELSE IF CHAR < F13 THEN CALL STORE(11);
00193   3               ELSE IF CHAR < GDP THEN CALL STORE(13);
00194   3               ELSE IF CHAR = GDP THEN CALL GOSDEPENDING;
00195   3               ELSE IF CHAR = BST THEN CALL BACKSSTUFF;
00196   3               ELSE IF CHAR = INT THEN CALL INITIALIZE;
00197   3               ELSE IF CHAR = TER THEN
00198   3               DO;
00199   3                   CALL PRINT(.'LOAD FINISHEDS$');
00200   4                   RETURN;
00201   4               END;
00202   3               ELSE IF CHAR = SCD THEN CALL STARTSCODE;
00203   3               ELSE DO;
00204   3                   IF CHAR <> OFFH THEN CALL PRINT(.'LOAD ERRCRS');
00205   4                   END;
00206   4           END;
00207   3       END BUILD;
00208   2
00209   1
00210   1
00211   1           /* PROGRAM EXECUTION STARTS HERE */
00212   1
00213   1       FCBSBYTE=0;
00214   1       CALL MOVE(.('CIN',0,0,0,0),FCB + 9,7);
00215   1       IF OPEN(FCB)=255 THEN
00216   1       DO;
00217   1           CALL PRINT(.'FILE NCT FOUND    $');
00218   2           GC TO BCCT;
00219   2       END;
00220   1       CALL NEXTSCHAR;
00221   1       CALL BUILD;
00222   1       CALL MOVE(.INTERFSFCB,FCB,33);
00223   1       IF OPEN(FCB)=255 THEN
00224   1       DO;
00225   1           CALL PRINT(.'INTERPRETER NCT FOUND    $');
00226   2           GC TO BCCT;
00227   2       END;
00228   1       CALL MOVE(READERSLOCATICN, 80H, 80H);
00229   1       GO TO 80H;
00230   1       EOF
```

163

```
00C01    1           /* THIS FRCGRAM TAKES THE CCOE CUTPUT FROM THE COBCL CCMPILER
00002    1              ANC CCNVERTS IT INTO A REAOABLE CUTPUT TO FACILITATE CEBUGGING */
00003    1
00C04    1
00C05    1  100H:            /* LOAD POINT */
00C06    1
00CC7    1  CECLARE
C0C08    1
C0C09    1  LIT              LITERALLY       'LITERALLY',
CCC10    1  8CCT             LIT             '0',
00011    1  8OCS             LIT             '5',
00C12    1  FCB              ACCRESS         INITIAL (5CH),
00C13    1  FCB$BYTE         BASED     FCB   BYTE,
00C14    1  I                BYTE,
00C15    1  AOCR             ACCRESS         INITIAL (100H),
C0C16    1  CHAR             BASEO     ADOR  BYTE,
C0C17    1  C$ACCR           BASED ADOR            ADORESS,
C0C18    1  BUFF$ENO         LIT             'OFFH',
C0C19    1  FILE$TYPE        CATA ('C','I','N');
C0C20    1
00C21    1  MCN1: PROCEDURE (F,A);
C0C22    2      CECLARE F BYTE, A ADORESS;
C0C23    2      GC TC BCCS;
C0C24    2  END MCN1;
C0C25    1
C0C26    1
00C27    1  MCN2: PROCECLRE (F,A) BYTE;
C0C28    2      OECLARE F BYTE, A AOORESS;
C0C29    2      GC TC BCCS;
CCC30    2  ENO MCN2;
C0C31    1
C0C32    1
C0033    1  PRINT$CHAR: PRCCECURE(CHAR);
C0C34    2      CECLARE CHAR BYTE;
C0C35    2      CALL MCN1(2,CHAR);
C0C36    2  END PRINT$CHAR;
C0C37    1
C0C38    1
C0C39    1  CRLF: PROCECLRE;
C0C40    2      CALL PRINT$CHAR(13);
C0C41    2      CALL PRINT$CHAR(10);
C0C42    2  ENC CRLF;
CCC43    1
J0C44    1
C0045    1  P: PRCCEOURE(ACC1);
C0C46    2      CECLARE ACCI ADDRESS, C BASED AO01 BYTE;
J0C47    2      CALL CRLF;
C0C48    2      CC I=0 TC 2;
C0C49    2          CALL PRINT$CHAR(C(I));
C0C50    3      ENO;
C0C51    2      CALL PRINT$CHAR(' ');
00C52    2  ENO P;
CCC53    1
00C54    1  GET$CHAR: PRCCECLRE BYTE;
C0C55    2      IF (ACCR:=ACCR + 1)>BUFF$ENO THEN
00C56    2      CC;
C0C57    2          IF MCN2(20,FCB)<>0 THEN
C0C58    3          CC;
C0C59    3              CALL P(.'ENO');
C0C60    4              CALL TIME(10);
00C61    4              CC TO BCOT;
CCC62    4          ENC;
C0C63    3          AOCR=6CH;
J0C64    3      END;
00C65    2      RETURN CHAR;
00C66    2  ENO GET$CHAR;
0CC67    1
CCC68    1
C0C69    1  OSCHAR: PRCCECLRE (OLTPUT$BYTE);
C0C70    2      CECLARE CLTFLT$BYTE BYTE;
00071    2      IF OUTPUT$YTE<IO THEN CALL PRINT$CHAR(OUTPUT$BYTE + 30h);
00C72    2      ELSE CALL PFINTSCHAR(OUTPUT$BYTE + 37H);
CCC73    2  ENO CSCHAR;
00C74    1
00C75    1
00C76    1  D: PRCCEOURE (CCLNT);
C0C77    2      CECLARE(CCLNT,J) AODRESS;
C0C78    2      CC J=1 TC CCLNT;
C0C79    2          CALL CSCHAR(SHR(GET$CHAR,4));
CCC80    3          CALL CSCHAR(CHAR AND OFH);
CCC81    3          CALL PRINT$CHAR(' ');
CCC82    3      ENC;
CCC83    2  END D;
CCC84    1
CCC85    1
0CC86    1  PRINT$REST: FRCCEOURE;
00C87    2      CECLARE
00C88    2      F2       LIT      '8',
CCC89    2      F3       LIT      '9',
CCC90    2      F4       LIT      '21',
CCC91    2      F5       LIT      '25',
CCC92    2      F6       LIT      '32',
C0C93    2      F7       LIT      '39',
CCC94    2      FS       LIT      '49',
0CC95    2      FIC      LIT      '54',
CCCS6    2      FI1      LIT      '6C',
0CC97    2      FI3      LIT      '61',
CCC98    2      CCP      LIT      '62',
CCC99    2      INT      LIT      '63',
C0100    2      BST      LIT      '64',
00101    2      TER      LIT      '65',
00102    2      SCO      LIT      '66';
C0103    2
```

164

```
00104    2          IF CHAR < F2 THEN RETURN;
00105    2          IF CHAR < F3 THEN DO; CALL D(1); RETURN; END;
00106    2          IF CHAR < F4 THEN DO; CALL D(2); RETURN; END;
00107    2          IF CHAR < F5 THEN DO; CALL D(3); RETURN; END;
00108    2          IF CHAR < F6 THEN DO; CALL D(4); RETURN; END;
00109    2          IF CHAR < F7 THEN DO; CALL D(5); RETURN; END;
00110    2          IF CHAR < F9 THEN DO; CALL D(6); RETURN; END;
00111    2          IF CHAR < F10 THEN DO; CALL D(8); RETURN; END;
00112    2          IF CHAR < F11 THEN DO; CALL D(9); RETURN; END;
00113    2          IF CHAR < F13 THEN DO; CALL D(10); RETURN; END;
00114    2          IF CHAR < GLP THEN DO; CALL D(12); RETURN; END;
00115    2          IF CHAR=CCF THEN DO; CALL D(1); CALL D(SHL(CHAR,1)+5); RETURN; END;
00116    2          IF CHAR=INT THEN DO; CALL D(3); CALL D(C$ADDR + 1); RETURN; END;
00117    2          IF CHAR=BST THEN DO; CALL D(4); RETURN; END;
00118    2          IF CHAR=TER THEN DO; CALL P(.'END'); GO TO BOOT; END;
00119    2          IF CHAR=SCC THEN DO; CALL D(2); RETURN; END;
00120    2          IF CHAR <> 0FFH THEN CALL P(.'XXX');
00121    2      END PRINT$REST;
00122    1
00123    1
00124    1          /* PROGRAM EXECUTION STARTS HERE */
00125    1
00126    1      FCB$BYTE=0;
00127    1      DO I=C TO 2;
00128    1          FCB$BYTE(I+9)=FILE$TYPE(I);
00129    2      END;
00130    1
00131    1      IF MON2(15,FCB)=255 THEN DO; CALL P(.'ZZZ'); GO TO BOOT; END;
00132    1
00133    1      DO WHILE 1;
00134    1          IF GET$CHAR <= 66 THEN DO CASE CHAR;
00135    2      ;       /* CASE 0 NOT USED */
00136    3              CALL P(.'ADD');
00137    3              CALL P(.'SUB');
00138    3              CALL P(.'MUL');
00139    3              CALL P(.'DIV');
00140    3              CALL P(.'NEG');
00141    3              CALL P(.'STP');
00142    3              CALL P(.'STI');
00143    3              CALL P(.'END');
00144    3              CALL P(.'RET');
00145    3              CALL P(.'CLS');
00146    3              CALL P(.'SER');
00147    3              CALL P(.'BRN');
00148    3              CALL P(.'CFN');
00149    3              CALL P(.'CP1');
00150    3              CALL P(.'CP2');
00151    3              CALL P(.'FGT');
00152    3              CALL P(.'RLT');
00153    3              CALL P(.'REC');
00154    3              CALL P(.'INV');
00155    3              CALL P(.'EQR');
00156    3              CALL P(.'ACC');
00157    3              CALL P(.'CIS');
00158    3              CALL P(.'STD');
00159    3              CALL P(.'LDI');
00160    3              CALL P(.'DEC');
00161    3              CALL P(.'STO');
00162    3              CALL P(.'ST1');
00163    3              CALL P(.'ST2');
00164    3              CALL P(.'ST3');
00165    3              CALL P(.'ST4');
00166    3              CALL P(.'ST5');
00167    3              CALL P(.'LOD');
00168    3              CALL P(.'LD1');
00169    3              CALL P(.'LD2');
00170    3              CALL P(.'LD3');
00171    3              CALL P(.'LD4');
00172    3              CALL P(.'LD4');
00173    3              CALL P(.'LD6');
00174    3              CALL P(.'FER');
00175    3              CALL P(.'CNU');
00176    3              CALL P(.'CNS');
00177    3              CALL P(.'CAL');
00178    3              CALL P(.'RWS');
00179    3              CALL P(.'CLS');
00180    3              CALL P(.'HDF');
00181    3              CALL P(.'WTF');
00182    3              CALL P(.'RVL');
00183    3              CALL P(.'WVL');
00184    3              CALL P(.'SCR');
00185    3              CALL P(.'SGT');
00186    3              CALL P(.'SLT');
00187    3              CALL P(.'SEQ');
00188    3              CALL P(.'MCV');
00189    3              CALL P(.'PRS');
00190    3              CALL P(.'WRS');
00191    3              CALL P(.'FRR');
00192    3              CALL P(.'WRR');
00193    3              CALL P(.'RWR');
00194    3              CALL P(.'DLR');
00195    3              CALL P(.'MED');
00196    3              CALL P(.'MNE');
00197    3              CALL P(.'GPD');
00198    3              CALL P(.'INT');
00199    3              CALL P(.'BST');
00200    3              CALL P(.'TFR');
00201    3              CALL P(.'SCD');
00202    3          END; /* CF CASE STATEMENT */
00203    2          CALL PRINT$REST;
00204    2      END; /* END CF DC WHILE */
00205    1      EOF...
```

```
01444    4
01445    4        /* RRR */
01446    4
01447    4                    CC;
01448    4                        CALL SET$RANDOM$POINTER;
01449    5                        CALL READ$TO$MEMORY;
01450    5                        CALL INC$PTR(9);
01451    5                    ENC;
01452    4
01453    4        /* WRR */
01454    4
01455    4                    CALL WRITE$RANDOM;
01456    4
01457    4        /* RWR */
01458    4
01459    4                    CALL WRITE$RANDOM;
01460    4
01461    4        /* CLR */
01462    4
01463    4                    CC;
01464    4                        CALL SET$RANDOM$POINTER;
01465    5                        CALL WRITE$ZERO$RECORD;
01466    5                        CALL INC$PTR(9);
01467    5                    ENC;
01468    4
01469    4        /* MED */
01470    4
01471    4                    CC;
01472    4                        CALL MOVE(C$ADDR(3),C$ADDR,C$ADDR(4));
01473    5                        BASE=C$ADDR(1);
01474    5                        HOLD=C$ADDR;
01475    5                        CTR=0;
01476    5                        DO WHILE (CTR<C$ADDR(1))AND(CTR<C$ADDR(4));
01477    5                            CALL CHECK$EDIT(H$BYTE);
01478    6                        END;
01479    5                        IF CTR < C$ADDR(4) THEN
01480    5                            CALL FILL(HOLD,C$ADDR(4)-CTR,' ');
01481    5                    ENC;
01482    4
01483    4        /* MNE */
01484    4
01485    4        ;
01486    4
01487    4        /* CCP */
01488    4
01489    4                    CC;
01490    4                        DECLARE OFFSET BYTE;
01491    5                        OFFSET=CONVERT$TO$HEX(C$ADDR(1),C$BYTE(1)-1);
01492    5                        IF OFFSET > C$BYTE + 1 THEN
01493    5                        DO;
01494    5                            CALL PRINT$ERROR('GD');
01495    6                            CALL INC$PTR(SHL(C$BYTE,1) + 6);
01496    6                        END;
01497    5                        ELSE PROGRAM$COUNTER=C$ADDR(OFFSET + 2);
01498    5                    ENC;
01499    4
01500    4              ENC; /* END OF CASE STATEMENT */
01501    3        END; /* END OF DO FOREVER */
01502    2    ENC EXECUTE;
01503    1
01504    1        /* * * * * * * * * * * PROGRAM EXECUTION STARTS HERE * * * * * * * */
01505    1
01506    1    BASE=CODE$START;
01507    1    PROGRAM$COUNTER=B$ADDR;
01508    1    CALL EXECUTE;
01509    1    EOF
```

166

```
UULUI   1
C3002   1           /* COBCL CCMPILER - PART 2 REACER */
COCO3   1
OCCO4   1           /* THIS FRCGRAM IS LOADED IN WITH THE PART 1 PROGRAM
00CO5   1           AND IS CALLEC WHEN PART 1 IS FINISHED.  THIS PROGRAM
00006   1           OPENS THE PART2.COM FILE THAT CONTAINS THE CODE FOR
CCCO7   1           PART 2 CF THE COMPILER, AND READS IT INTO CCRE. AT
COOO8   1           THE END CF THE READ CPERATICN, CCNTROL IS PASSED TC
00009   1           THE SECCND PART FROGRAM.                          */
COC10   1
COC11   1
00012   1   3100H:   /* LCAC POINT */
COC13   1
COC14   1   DECLARE
COC15   1
00C16   1   BCCT     LITERALLY 'OH',
00C17   1   BDCS     LITERALLY '5F',   /* ENTRY TC THE OPERATING SYSTEM */
COO18   1   START    LITERALLY 'ICOH',   /* STARTING LOCATICN FOR PASS 2 */
COC19   1   FCB (33) BYTE INITIAL(0,'PASS2   COM',0,0,0,0),
COC2O   1   LASTCMA  ADCRESS  INITIAL(2480H), /* 8O LESS THAN MEMCRY */
00C21   1   I        ADCRESS;
COC22   1
00C23   1   MCNA: PROCECLRE(F,A);
00C24   2       DECLARE F BYTE, A ADDRESS;
COC25   2       GC TO BCCS;
COC26   2   END MCNA;
00C27   1
COC28   1   MCNB: FROCECLRE(F,A)BYTE;
COC29   2       DECLARE F BYTE, A ADDRESS;
COC3O   2       GC TC BOCS;
COC31   2   END MCNB;
COC32   1
00C33   1   ERRCR: FROCECURE(CODE);
COC34   2       DECLARE CCDE ADDRESS;
COC35   2       CALL MCNA(2,(FIGH(CCDE)));
00C36   2       CALL MCNA(2,(LCW(CODE)));
00C37   2       CALL TIME(IC);
00C38   2       GO TC BCCT;
COC39   2   END ERROR;
COC40   1
00C41   1       /* OPEN PASS2.CCM */
00C42   1   IF MONE(15,.FCB)=255 THEN CALL ERRCR('O2');
00C43   1       /* READ IN FILE */
00C44   1   CO I=ICOH TO LASTCMA BY 80H;
COC45   1       CALL MCNA(26,I);  /* SET DMA */
COC46   2       IF MONB(2C,.FCB)<>O THEN CALL ERRCR('R2');
00C47   2   END;
COO48   1   CALL MCNA(26,8CH); /* RESET DMA */
00C49   1   GC TC START;
COC50   1   EOF



00CO1   1           /* COBOL CCMPILER - INTERP REACER */
00CO2   1
00CO3   1           /* THIS FRCGRAM IS CALLEO BY THE BUILD PROGRAM AFTER
00004   1           CBLINT.CCM HAS BEEN OPENEO, AND READS THE CODE INTC MEMCRY
COCO5   1           */
COCO6   1
00CC7   1
OCCO8   1   80H:   /* LCAD FCINT */
COCO9   1
COC1O   1   DECLARE
00C11   1
00C12   1   BCCT     LITERALLY 'OH',
00C13   1   BCCS     LITERALLY '5F',   /* ENTRY TO THE OPERATING SYSTEM */
COO14   1   START    LITERALLY 'ICOH',   /* STARTING LCCATICN FOR PASS 2 */
COC15   1   LASTOMA  ADCRESS  INITIAL(1E80H), /* 8O LESS THAN MEMCRY */
COC16   1   I        ADCRESS;
00C17   1
00C18   1   MCNA: PROCECLRE(F,A);
COC19   2       DECLARE F BYTE, A ADDRESS;
COC2O   2       GC TC BCCS;
COC21   2   END MCNA;
COC22   1
00C23   1   MCNB: FROCECLRE(F,A)BYTE;
COC24   2       DECLARE F BYTE, A ADCRESS;
COC25   2       GC TC BCCS;
COC26   2   END MCNB;
00C27   1
COC28   1   OO I=ICOH TC LASTOMA BY 80H;
COC29   1       CALL MCNA(26,I);  /* SET DMA */
COC3O   2       IF MONB(20,SCH)<>O THEN GO TO BOOT;
00031   2   END;
00C32   1   GC TO START;
OOC33   1   ECF
```

LIST OF REFERENCES

1. American National Standards Institute, COBOL Standard, ANSI X3.23-1974.

2. Aho, A. V. and S. C. Johnson, LR Parsing, Computing Surveys, Vol. 6 No. 2, June 1974.

3. Bauer, F. L. and J. Eickel, editors, Compiler Construction - An Advanced Course, Lecture notes is Computer Science, Springer-Verlag, New York 1976.

4. Digital Research, An Introduction to CP/M Features and Facilities, 1976.

5. Digital Research, CP/M Interface Guide, 1976.

6. Eubanks, Gordon E. Jr. A Microprocessor Implementation of Extended Basic, Masters Thesis, Naval Postgraduate School, December 1976.

7. Intel Corporation, 8008 and 8080 PL/M Programming Manual, 1975.

8. Intel Corporation, 8080 Simulator Software Package, 1974.

9. Knuth, Donald E. On the Translation of Languages from Left to Right, Information and Control Vol. 8, No. 6, 1965.

10. Software Development Division, ADPE Selection Office, Department of the Navy, HYPO-COBOL, April 1975.

11. Strutynski, Kathryn B. Information on the CP/M Interface Simulator, internally distributed technical note.

12. University of Toronto, Computer Systems Research Group

Technical Report CSRG-2, "An Efficient LALR Parser Generator," by W. R. Lalonge, April 1971.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Documentation Center   2
   Cameron Station
   Alexandria, Virginia 22314

2. Library, Code 0212   2
   Naval Postgraduate School
   Monterey, California 93940

3. Department Chairman, Code 52   1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93940

4. Assoc Professor G. A. Kildall, Code 52Kd   1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93940

5. Lt L. V. Rich, Code 52Ks   1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93940

6. ADPE Selection Office   1
   Department of the Navy
   Washington, D. C. 20376

7. Capt A. S. Craig, USMC   1
   611 Canyon Drive,
   Springville, Utan 84663